1994-12

# A mine search algorithm for the Naval Postgraduate School Autonomous Underwater Vehicle

Rodrigues Neto, Jose Augusto

Monterey, California. Naval Postgraduate School

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California




# THESIS

A MINE SEARCH ALGORITHM FOR THE NAVAL POSTGRADUATE SCHOOL AUTONOMOUS UNDERWATER VEHICLE

by

Jose Augusto Rodrigues Neto

December 1994

Thesis Advisors: Gordon H. Bradley
Robert B. McGhee

Approved for public release; distribution is unlimited

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| 1. AGENCY USE ONLY (Leave Blank) | 2. REPORT DATE | 3. REPORT TYPE AND DATES COVERED |
|---|---|---|
| | December 1994 | Master's Thesis |

**4. TITLE AND SUBTITLE**
A Mine Search Algorithm for the Naval Postgraduate School Autonomous Underwater Vehicle (U)

**5. FUNDING NUMBERS**

**6. AUTHOR(S)**
Rodrigues Neto, Jose Augusto

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**
Naval Postgraduate School
Monterey, CA 93943-5000

**8. PERFORMING ORGANIZATION REPORT NUMBER**

**9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

**10. SPONSORING/ MONITORING AGENCY REPORT NUMBER**

**11. SUPPLEMENTARY NOTES**
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

**12a. DISTRIBUTION / AVAILABILITY STATEMENT**
Approved for public release; distribution is unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT** *(Maximum 200 words)*

This thesis develops, implements and tests a mine search algorithm for the Naval Postgraduate School Autonomous Underwater Vehicle (Phoenix). The vehicle is 72 inches long and displaces 400 pounds. Its maneuvers are performed using two propellers and four thrusters. It contains two embedded computer systems. The algorithm directs the autonomous search of a specified area mapping all obstacles and computing an estimate of the cumulative probability of detection. The algorithm uses no prior knowledge of the terrain or the location of mines. The algorithm, which is written in Lisp, can execute on the vehicle's computer systems. Along with the search and mapping capabilities, the algorithm executes obstacle avoidance. The algorithm is tested in several simulated scenarios with different placement of mines and obstacles; the amount of resources used and the fraction of the area searched is computed. A similar algorithm that uses hill-climbing search is implemented for comparison. In all cases, the newly developed algorithm performed equal or better than the one that uses hill-climbing.

| 14. SUBJECT TERMS | 15. NUMBER OF PAGES |
|---|---|
| Autonomous vehicle, mapping, mine, mine search, obstacle avoidance, robot | 103 |
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT | 18. SECURITY CLASSIFICATION OF THIS PAGE | 19. SECURITY CLASSIFICATION OF ABSTRACT | 20. LIMITATION OF ABSTRACT |
|---|---|---|---|
| Unclassified | Unclassified | Unclassified | UL |

**Approved for public release; distribution is unlimited**

# A MINE SEARCH ALGORITHM FOR THE NAVAL POSTGRADUATE SCHOOL AUTONOMOUS UNDERWATER VEHICLE

by

Jose Augusto Rodrigues Neto
Lieutenant, Brazilian Navy
B.S., Brazilian Naval Academy, 1982

Submitted in partial fulfillment of the
requirements for the degrees of

**MASTER OF SCIENCE IN OPERATIONS RESEARCH**

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

NAVAL POSTGRADUATE SCHOOL

**December 1994**



Author: [redacted]
Jose Augusto Rodrigues Neto

Approved By: [redacted]
Gordon H. Bradley, Thesis Advisor

[redacted]
Robert B. McGhee, Thesis Advisor

[redacted]
James N. Eagle, Second Reader

[redacted]
Ted Lewis, Chairman,
Department of Computer Science

[redacted]
Peter Purdue, Chairman,
Department of Operations Research

**ABSTRACT**

This thesis develops, implements and tests a mine search algorithm for the Naval Postgraduate School Autonomous Underwater Vehicle (Phoenix). The vehicle is 72 inches long and displaces 400 pounds. Its maneuvers are performed using two propellers and four thrusters. It contains two embedded computer systems. The algorithm directs the autonomous search of a specified area mapping all obstacles and computing an estimate of the fraction of area searched. The algorithm uses no prior knowledge of the terrain or the location of mines. The algorithm, which is written in Lisp, can execute on the vehicle's computer systems. Along with the search and mapping capabilities, the algorithm executes obstacle avoidance. The algorithm is tested in several simulated scenarios with different placement of mines and obstacles; the amount of resources used and the fraction of area searched is computed. A similar algorithm that uses hill-climbing search is implemented for comparison. In all cases, the newly developed algorithm performed equal or better than the one that uses hill-climbing.

TABLE OF CONTENTS

**ACKNOWLEDGEMENTS**

I would never be able to pursue my path without a strong belief in God. Whatever form it took, it was always with me.

The following persons have also supported me along this arduous path and I'll be forever grateful.

Don Brutzman, my instructor and friend, introduced me to the NPS AUV Project. With his unique ability to make everything simple, he was the first one to believe my goals were achievable and always managed to convince me. I will never forget your friendship.

Dr. Gordon Bradley and Dr. Robert McGhee, my advisors, guided me in a very special way. Your capability of understanding my needs, keeping me on the right track, were stimulating. I can only impute this to your experience and your intellectual capacity.

Dr. Anthony Healey, chairman of the Mechanical Engineering Department, conducted, with Dr. McGhee, the NPS AUV Project with mastery. Your willingness to help was always noted.

Dr. Alan Washburn, my professor and squash partner, could always give me a word of guidance on the so difficult search subject, even when recovering from a loss in the court.

Dr. James Eagle, my professor, introduced me to the search field.

David Marco, Ph.D. student was a driving force in the Project.

Charles Daniels and his family, our sponsors, made our life in a foreign country a gratifying experience.

To my Project colleagues, my classmates, the NPS staff and all the people that help me in one way or another, thank you.

This thesis is dedicated to the following special persons:

My parents, responsible for most of what I was able to accomplish. Their eternal dedication to us is beyond comprehension.

My wife, Angela, who supported me in the most difficult moments and believed in me when I was unable to do so.

My son, Helio, as his name describes, my sun. Your spiritual strength and your patience with your father during these last two years were admirable for a four year old boy.

My daughter, Luiza, my charming moon. Your happiness, and joy of life has always feed me. I will always be intrigued by how easily you understood that "Dad has to study".

# EXECUTIVE SUMMARY

## A. BACKGROUND

The advancement of technology is allowing the substitution of machines for men in several fields. The military is certainly one of them. Due to its nature, filled with so-called "dirty jobs", the military is the perfect place for employment of robots. Mine warfare is a specially suitable environment for development and use of this technology. Mine search gains a new perspective with the use of Autonomous Underwater Vehicles (AUV's). The absence of knowledge about the environment, the necessity of performing obstacle avoidance, the short range of the sensors, the object recognition problem, together with the ability to maneuver in three dimensions make the mine search through AUV's an interesting and challenging problem.

The NPS AUV project was begun in 1987 as a joint effort of the Mechanical Engineering, Computer Science and Electrical Engineering departments. It began with the sponsorship of the Naval Surface Weapons System (NSWC). With the evolution of the project, other agencies joined the group of sponsors, supporting different aspects of the research. The Undersea Warfare Curriculum and the Operations Research Department contribute to

the project, either by providing financial support or students to work in the NPS AUV group. The project encompasses research in areas of mechanical engineering, control systems, artificial intelligence, and computer communications and networks.

## B. DEVELOPMENT

This thesis develops and implements a mine search algorithm for the Naval Postgraduate School Autonomous Underwater Vehicle (Phoenix). To give the robot decision making ability the algorithm uses artificial intelligence techniques. The algorithm is based on a high level "raster-scan" strategy combined with a sweeping heuristic and hill-climbing procedures. This thesis also establishes measures of effectiveness to evaluate the algorithm and possible future developments.

## C. ACCOMPLISHMENTS

The algorithm directs the autonomous search of a specified area, without prior knowledge of its characteristics, mapping all obstacles and computing an estimate of the fraction of area searched. It is written in Lisp, an AI language, and proved to be fast enough to drive the vehicle. Along with the search and mapping capabilities, the algorithm performs obstacle avoidance. The algorithm is tested in several simulated scenarios with different placement of mines and obstacles; the amount of resources used and the fraction of area searched is computed. A

similar algorithm that uses hill-climbing search is implemented for comparison. In all cases, the newly developed algorithm performed as well or better than the one that uses hill-climbing. Experimental results show that an average fraction of the area searched of 0.7 is achieved using this algorithm.

# I. BACKGROUND

## A. INTRODUCTION

The advancement of technology is allowing the substitution of machines for men in several fields. The military is certainly one of them. Due to its nature, filled with so-called "dirty jobs", the military is the perfect place for employment of robots. Mine warfare is a specially suitable environment for development and use of this technology.

Naval search problems have received special attention since the creation of the Anti-Submarine Warfare Operations Research Group (ASWORG), later called Operations Evaluation Group (OEG), during World War II [Ref. 1]. The great threat presented by the German Navy, particularly by its submarines, led the United States into scientifically studying the search problem. Those studies provided the basis for further development of such problems.

Mine search constitutes a particularly delicate problem. Despite the risk that the actual conduct of the search presents to the searcher, or sweeper, the results of a sweeping can dramatically change the course of operations.

## B. THE NAVAL POSTGRADUATE SCHOOL AUTONOMOUS UNDERWATER VEHICLE (PHOENIX) PROJECT

The NPS AUV project was begun in 1987 as a joint effort of the Mechanical Engineering, Computer Science and Electrical Engineering departments. It began with the sponsorship of the Naval Surface Weapons System (NSWC). The first vehicle built was based on the Navy's Swimmer Delivery Vehicle (SDV) [Ref. 2]. With the evolution of the project, other agencies joined the group of sponsors, supporting different aspects of the research. The Undersea Warfare Curriculum and the Operations Research Department contribute to the project, either by providing financial support or students to work in the NPS AUV group. The project encompasses research in areas of mechanical engineering, control systems, artificial intelligence, and computer communications and networks.

The present NPS AUV, named *Phoenix*, has a total length of 72 inches and displaces about 400 pounds. It is propelled by two electrical motors and has four transverse thrusters, two horizontal and two vertical, that give it a unique maneuverability capability. Attitude control is provided by four horizontal and four vertical fins in conjunction with the thrusters. It has two embedded computer systems. One performs the vehicle's hardware control, known as the Execution Level. The other provides what is called the Tactical and Strategic Level. The Tactical Level is

the intermediate level control of the AUV, as further described. Two sonars are used. The ST725 is a scanning sonar that operates at 725 Khz and has an approximate range of 40 m. It has a beam 1 degree wide by 24 degrees vertically, with a resolution of 1 cm. It is used for area survey. The ST1000 operates at 1000 Khz, and has a 1 degree conical beam. It is utilized as a profiling sonar. Both sonars are 360 degrees steerable. A schematic drawing of the NPS AUV is shown in Figure 1.

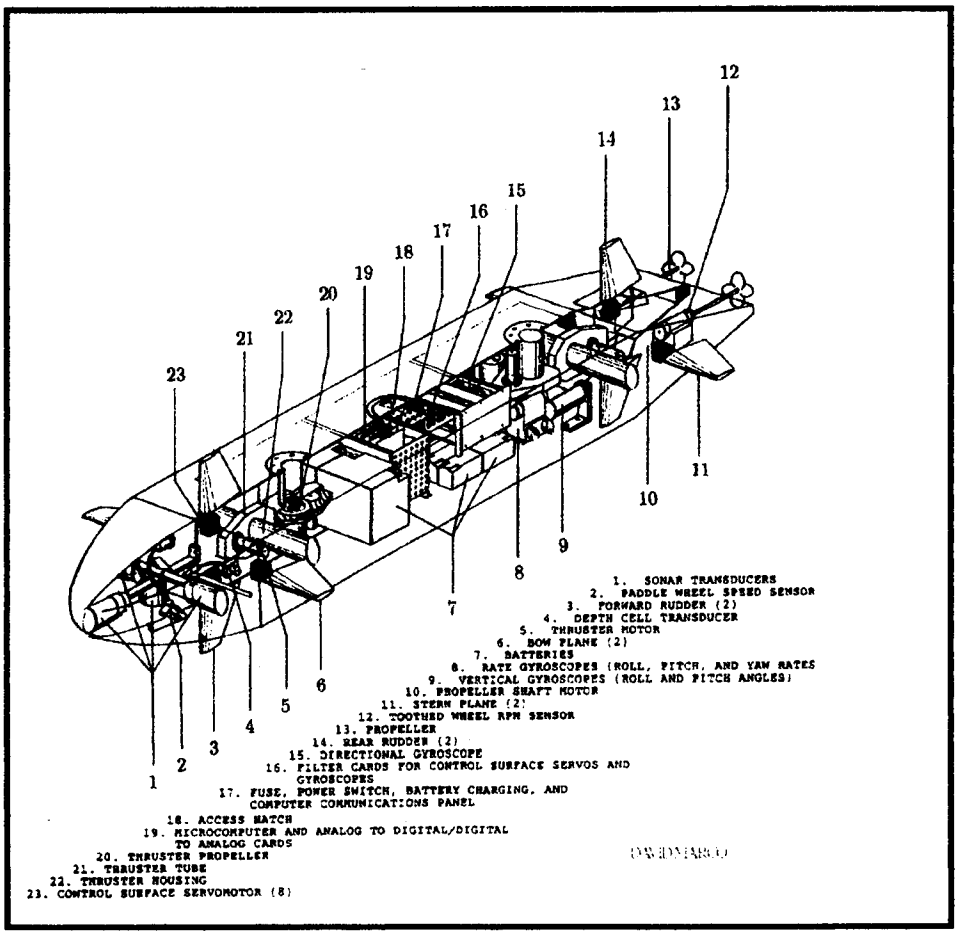


Figure 1: The NPS AUV schematic drawing

An underwater virtual world and an AUV simulator are also being developed as part of the project. They allow the evaluation of the robot and most of its components without putting it at risk [Ref. 3].

## C. THESIS OBJECTIVES

The purpose of this thesis is to develop and implement, using an artificial intelligence language, a mine search algorithm for an autonomous underwater vehicle. Since it will be executed by a robot, the main requirement for the algorithm is to be realistic and simple. The algorithm has to consider the three-dimension movement capability of the robot in the absence of prior knowledge of the environment. It has to provide the robot with obstacle avoidance. As an output, the algorithm shall provide an estimate of the attained fraction of area searched and a map of the searched volume. This thesis will also explain how the current algorithm interfaces with the robot.

## D. THESIS ORGANIZATION

This thesis is organized in six chapters. Chapter II provides detailed information on the AUV software organization, general mine search considerations and a summary of previous related work. Chapter III defines the problem and the assumptions made to obtain a solution. Chapter IV describes the solution and explains the new algorithm. Chapter V addresses typical

environment configurations and provides experimental results. Chapter VI discusses the algorithm and establishes foundations for further research on the topic.

## E. SUMMARY

An AUV is possibly the best way to perform mine search. Its autonomous characteristic is welcome as a way to reduce human risk significantly. The effective use of a robot poses intriguing questions whose answers have ramifications in artificial intelligence, machine learning, optimization and other related fields.

This chapter gives a brief introduction to the NPS AUV project and explains its relation to mine search and to this thesis. It includes a description of the thesis objectives and organization.

# II. PREVIOUS WORK

## A. INTRODUCTION

Mines have been around since the American Civil War. Mines dating from the WW I are still in use and can be very effective [Ref. 4].

Two basic mine countermeasures are mine sweeping and mine hunting. Although those procedures have improved during the years, they are limited by the fact that the executing agent is a ship, with the exception of some helicopters used for mine hunting. The current development of autonomous robots is gradually changing this scenario.

## B. CONTROL ARCHITECTURE

The control of a robot naturally demands different levels of abstraction. The NPS AUV uses a software architecture called the Rational Behavior Model (RBM). [Ref. 7]

The RBM defines three levels of abstraction. The Strategic level is the highest level of abstraction and is designed to guide the overall vehicle behavior using only symbolic computation. This level has no time constraints and communicates asynchronously with the next level, the Tactical level.

The Tactical level is the interface between the highest or Strategic level and the lowest or Execution level. The Tactical level is responsible for implementing the general behaviors defined by the Strategic level, issuing specific commands to the Execution level, and returning the pertinent information to the upper level. The most common command issued to the lower level is a setpoint, e.g., the next waypoint. It is also responsible for the analysis of the data supplied by the Execution level.

The Execution level is responsible for directly controlling the robot hardware. It performs only numeric processing and hosts all hard real-time processes. It controls the vehicle stability and has provisions to be the final safety agent. It overrides any command that can endanger the vehicle.

The simplest and most common used analogy for the different levels is a real submarine organization. The Strategic level corresponds to the Commanding Officer. The Tactical level is equivalent to the OOD and his assistants. The Execution level is represented by the execution agents on the submarine, e.g., the helmsman and other crew members.

### C. SEARCH AND MINEFIELD CONSIDERATIONS

The theoretical foundations for search theory were established by Koopman [Ref. 1]. Koopman's work was later expanded by many authors and received special attention from Stone

[Ref. 8]. References [9,10] are surveys of works in search theory and contain extensive bibliographies.

Minefield search has received little attention from the search community. The areas that have received more attention are minefield planning and simulation.

Two good search strategies are defined by Washburn for multiple stationary target search [Ref. 11]. In both cases though, it is assumed that the searcher has to actually visit all detected targets, which is not the AUV case. It also makes no provision for obstacle avoidance, most likely because it was conceived for surface ships.

The search performed by the AUV takes place in a volume and not in an area as most models define. There is not a provision in any theoretical model that accounts for unpredicted obstacles in the search path, not even for known obstacles. In addition, the search theory literature generally assumes that when the target is found it is immediately recognized by the searcher.

In robotics, most of the attention is directed to obstacle avoidance and dynamic path planning in unknown terrain. Lumelsky defines very good solutions to the dynamic path planning problem [Ref. 12]. Another approach to the same problem is defined by Krogh and Feng [Ref. 13]. A mine avoidance problem is solved by Hyland and Taylor [Ref. 6]. A survey of motion planning

algorithms is found in reference [14]. Caddell defines a method for three-dimensional path planning [Ref. 15].

The limitation of the present object recognition techniques and the inability of the AUV to disarm or destroy a mine calls for a map generating capability. The generated map is evaluated by the required specialists and used to direct the neutralization efforts. A simplified occupancy grid approach is used, with no uncertainty [Ref. 16].

Compton has a fairly complete study of the problem and this thesis has certainly benefited from that [Ref. 2]. However, certain assumptions used in his model are not used here.

### D. SUMMARY

The AUV is a sound representation of an emerging technology. The RBM software model is an attempt to cope with the many distinct problems concerning the implementation and control of a robot.

Mine search gains a new perspective with the use of AUV. The absence of knowledge about the environment, the necessity of performing obstacle avoidance, the short range of the sensors, the object recognition problem, together with the ability to maneuver in three dimensions make the mine search through AUV's an interesting and challenging problem.

# III. PROBLEM DEFINITION

## A. INTRODUCTION

The mine search algorithm is intended to be realistic and simple as stated in the Thesis Objectives section. These two principles bound the way assumptions are made about the real world. When a conflict between the principles occurs, the simple solution is chosen. This is done to avoid any decision that may compromise the algorithm's effectiveness in a real operation.

## B. MINE SEARCH

The search will be defined considering the type of target and environment characteristics. Although differences may exist regarding minefield characteristics, due to different mining objectives and type of missions to be supported, these are not taken into account.

The only data provided to the algorithm are the volume coordinates. The parameters needed to determine the sonar performance under the actual specific conditions and the available resources, i.e., maximum time available for the mission, are used to tailor the algorithm for the specific search conditions. No

prior knowledge of the terrain or the location of the mines is assumed.

The algorithm can be adjusted for the actual parameters and will discover the characteristics of the terrain as the search progresses.

The position of the mines and a map of the terrain will be the result of the search. The total fraction of the area searched will be calculated as well as the total amount of resource spent during the search.

### C. SEARCH SPACE DEFINITION

To compare different search strategies a basic search space will be defined. The search space will be a rectangular prism, described as a three-dimensional matrix. The vehicle will only be allowed to be in one cell at a time. If there are no obstacles the vehicle will have 26 neighbor cells that it can move to from its present position. These assumptions were also used by Compton [Ref. 2].

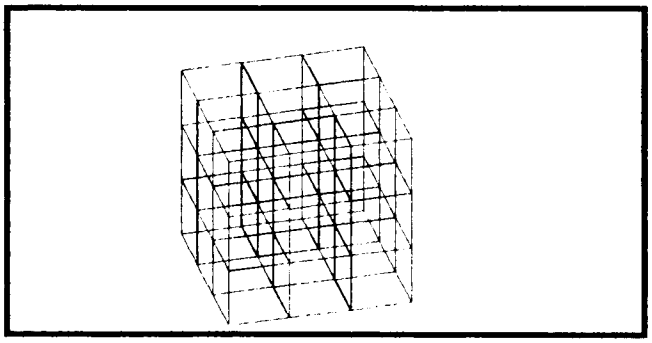

Figure 2: A 27-cell block

Using these basic definitions several scenarios with distinct obstacles and mine displacements will be used to allow a fair evaluation of the heuristic.

### D. OBSTACLE AVOIDANCE

The AUV Execution Level can perform obstacle avoidance by itself. That ability is provided as a safety measure and although it protects the integrity of the robot it does not contribute to the search itself. The Execution Level avoidance maneuver will not be used by the mine search algorithm. The algorithm will provide the AUV with obstacle avoidance behavior that will contribute to the search being conducted.

Although it is part of the vehicle architecture, we use no object recognition module. The system is easily modified to use any recognition algorithm that uses the profiling sonar. The existence of such an object would certainly help in the search.

All the obstacles will be mapped. If the object recognition module was available, obstacles could be mapped using different representation depending on their characteristics. The algorithm will give the vehicle the ability to determine the precise location and the approximate size of the object.

Generally all the areas that can be mined are mapped. Although the actual obstacles may not be represented in the charts, the depth information is reliable. The largest depth

within the search area will be given to the algorithm. This will be called Base Depth as shown below. This information is taken from the area coordinates given to the algorithm. An example of an (artificial) bottom contour map is provided by Fig. 3.

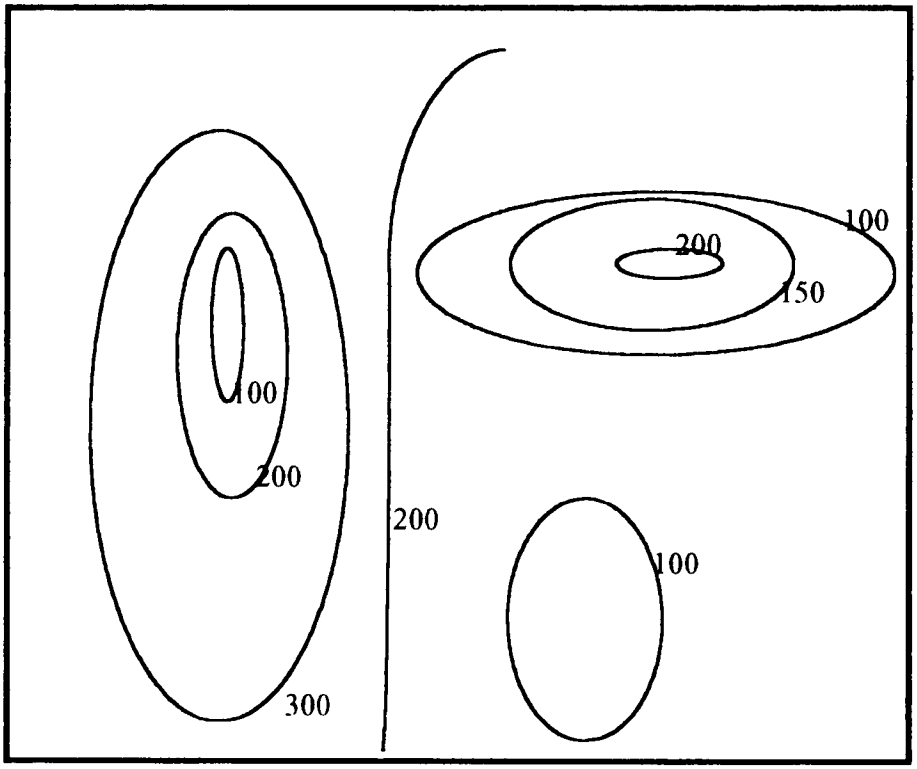


Figure 3: Bottom contour lines example (Base Depth = 300 ft)

**E. AUTONOMOUS CONTROL**

The most important feature of the algorithm is its capacity to perform a search mission autonomously. Due to this necessity the algorithm will behave in a way that reinforces the following directives:

- The vehicle shall above all avoid getting into an unrecoverable situation.
- Reliable information shall be given to the user when the mission is completed.
- The available resources shall be used always trying to improve the effectiveness of the search.

Considering that the vehicle will be planning and replanning its own path, it will have basic maneuvering strategies defined. The search will be an exhaustive search and will follow a "raster scan" pattern [Ref. 11]. Since the search space is defined in three dimensions, two possibilities arise:

- Horizontal Layering
- Vertical Layering

The earlier is the simpler approach. The search is conducted as a horizontal ladder pattern. Obstacle avoidance is performed usually through lateral turns. After a whole horizontal layer is searched, the vehicle goes deeper, proceeding to the next layer. This is frequently called "Cake Layering."

Vertical layering although not as simple for the AUV, due to the kind of maneuvers that will be required, has some advantages, as will be discussed below. In this case the obstacle avoidance maneuvers will usually be achieved by vertical changes to the course. The search space will then be divided in vertical slices

and normally a slice change will be performed only after a whole vertical slice has been searched.

Those strategies will determine the behavior of the vehicle in different maneuvering or decision making situations.

## F. RESOURCE OPTIMIZATION CONSIDERATIONS

The lack of knowledge about the search volume limits the development of an optimal policy. The position of the obstacles and mines can not be anticipated, and so replanning may be needed at each step of the search. The location of obstacles and mines is so critical that it may cause an early termination of the search due to a shortage of resources.

The possibility of an early termination of a search makes the vertical layering algorithm a better choice. If the horizontal layering method is used and the search is not completed, the whole volume will remain interdicted. Using the vertical layering though, whenever the search is halted, part of the volume will be already clean.

The need to have a simple and realistic algorithm, added to the real-time characteristics of such a problem, reduces the possibility of finding an optimal policy. Obstacles are found during the execution, and at each step new constraints can be added which force the AUV to replan the search.

In the AUV case, the resource to be saved is the battery. In a later stage of the project, the Tactical Level will receive real information about the situation of the battery set and use this information to replan the search path. While this feature is not yet implemented on the Phoenix vehicle, the measure of effectiveness (MOE) to be used will be the amount of resource needed to cover the whole search volume.

For the purpose of this simulation the MOE associated with the resource will be a Unit of Resource or UR. Different amounts of UR will be related to each specific maneuver that can be performed by the vehicle, as defined below:

- Level, straight move: 1 UR
- Level, 45° turn: 1.1 UR
- Level, 90° turn: 1.2 UR
- Level, 135° turn: 1.3 UR
- Level, 180° turn: 1.4 UR
- Level, lateral move: 1.5 UR
- Level, reverse move: 2.0 UR
- Vertical depth change: 2.0 UR

Any non-vertical depth changes just add 0.3 UR to the values above.

## G. FRACTION OF AREA SEARCHED

The fraction of area searched will be calculated using the "cookie-cutter" assumption [Ref. 5]. The sonar range can be varied to investigate the performance of the strategies in different scenarios.

The sonar detection region will be a sphere centered at the nose of the vehicle, as shown in Figure 4. Considering the "cookie-cutter" assumption, the fraction of area searched will be the number of cells touched by the sphere divided by the total number of cells in the search volume.

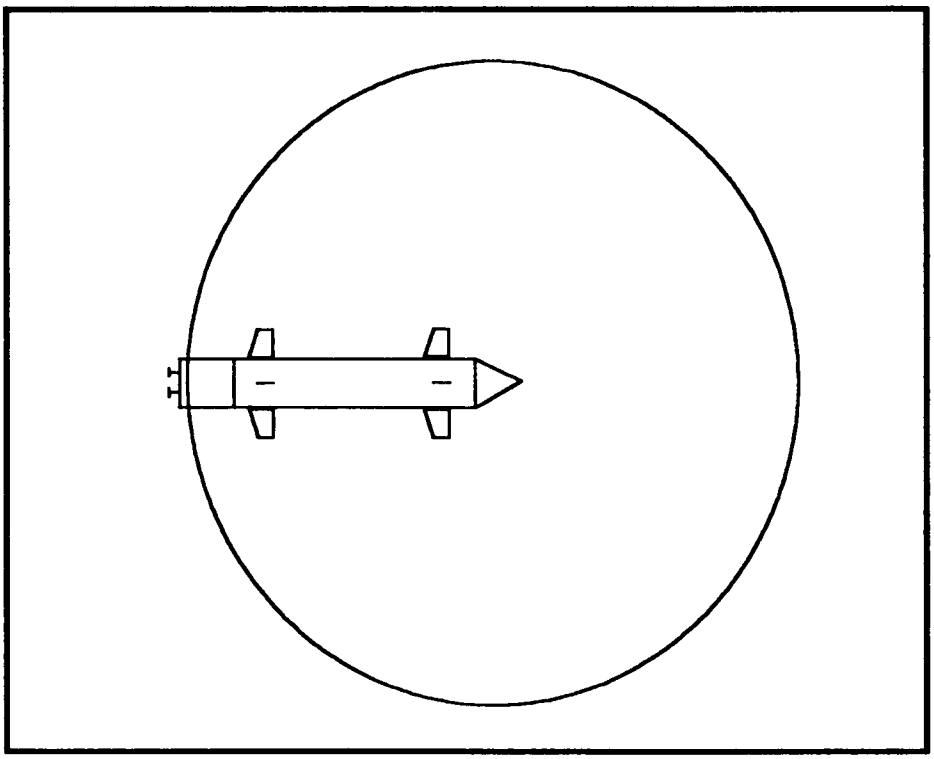

Figure 4 : AUV sonar profile

For the purpose of this thesis, the sphere defined by the sonar range is assumed to contain a cube composed of 27 cells.

## H. SUMMARY

The mine search is directed to moored mines. The search space is defined as a rectangular prism, divided into rectangular volume elements (or voxels as defined in Hyland and Taylor's work) [Ref. 6]. The whole volume will be called *search area* or *area*. The user provides just the area coordinates, maximum depth and sonar range. There is no available map of the terrain. Obstacle avoidance has to be performed by the algorithm. A post-search map of the volume has to be generated.

Costs are associated with each type of maneuver performed by the AUV. A fraction of area searched is accessed at the end of the search and is used as a MOE together with the total cost of the mission.

# IV. ALGORITHM DEVELOPMENT

## A. INTRODUCTION

One might think that it would be sufficient to establish a goal at the other side of the area and perform some kind of shortest path with obstacle avoidance. This is not the case. It is easy to see that in the case where a large obstacle blocks the originally planned path, the avoidance maneuver would leave a large area uncovered (Fig. 5). This is unacceptable for a mine search. To overcome this difficulty, sub-goals are generated by the obstacles along the path to force the AUV to fill the gaps. To plan the new path a customized procedure is used. It is developed with the intent to mimic a human faced with the same problem.

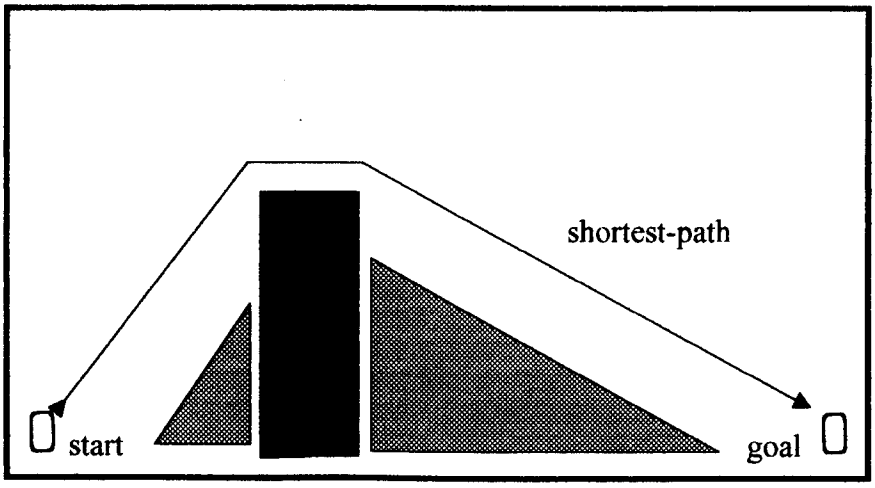


Figure 5: Result of a shortest-path procedure

### B. SEARCH SPACE MODELING

The form of the search space depends directly on the values supplied by the user. With the area coordinates, a rectangular prism is defined, bounded horizontally by those coordinates and vertically by the surface and the maximum depth, also supplied by the user. On the basis of the sonar range, the volume is divided in cubic cells. In this specific case the cell's dimension will be 20% of the range. This value is used to ensure that 27 cells will be contained by sonar sphere. For the AUV this will imply a cell with side length of approximately 20 ft.

### C. OBSTACLE MAPPING

Obstacles are mapped as sets of cells. If when surveying the area the AUV discovers a new obstacle it marks the cell that contains the obstacle coordinate as unsafe. Without using the profiling sonar, different cell sizes determine the definition of the map. The use of the appropriate tool, e.g., the profiling sonar to allow a reasonable definition of the obstacle, could provide a better terrain map to the user.

### D. MULTIPLE LAYERS SEARCH

The ability to maneuver in three dimensions enhances the capability of the vehicle for mine search. On the other hand maneuvering in three dimensions makes the total area coverage a

much harder problem, even for humans. To simplify the search the area is divided into vertical slices or layers where the AUV maneuvers using a two-dimensional behavior. The robot performs a complete search of each slice and then moves to the next.

## E. SUGGESTED HEURISTICS

### 1.Definitions

The basic algorithm will be defined using the vertical slices approach. Within a slice, the base depth for each search depth is called the level. The main sub-goal is the one defined at the opposite side of the volume, at the same level as the starting point. Consider an XYZ coordinates system, where the x-axis is the horizontal one along the slice, the z-axis the vertical one and the y-axis the horizontal axis that crosses the slices. If the starting point is (x1, y1, z1) the main sub-goal would be (x1+leg's length, y1, z1). Using that coordinate system, three flags are defined to guide the behavior of the AUV during the search. The x-motion can assume the values North or South. The y-motion can assume the values East or West. Finally the z-motion can assume the values Up or Down. Those flags will be used to bind the path planning and the movement of the vehicle when a decision point is reached. A decision point is a point where some event requires the replanning of the path. The agenda is a stack where the passive sub-goals are kept until they are reactivated,

and that has on top the active sub-goal. A cell can assume four states: unknown, safe, visited, and unsafe. A visited cell is a safe cell that has been occupied by the AUV. For planning purposes, unknown and safe cells are considered the same.

**2.The Procedure**

The vehicle starts the search in a corner of the area, preferably one at the surface. In the case it is deployed in a position non-coincident with a corner, it can maneuver to position itself in a corner. When it starts the search the planner defines the level's main sub-goal at the opposite side of the area, in the same slice and at the same depth. A sweeping path is defined to the main sub-goal. The motion flags are set accordingly. The AUV starts its movement based on the previously calculated sweeping path. When a cell in the path is found unsafe, the planner defines a sub-goal right after that cell, at the same level and slice (same z and y coordinates) of the unsafe cell, stores the previous sub-goal on the agenda and calculates, using the sweeping heuristic, the sweeping path for the new goal. The AUV then continues to the new cell, following that new sweeping path. Figure 6 shows an example of a search being conducted by the AUV. The original path was planned following the bottom line. When the AUV found an obstacle a sub-goal was generated and a new path was planned.

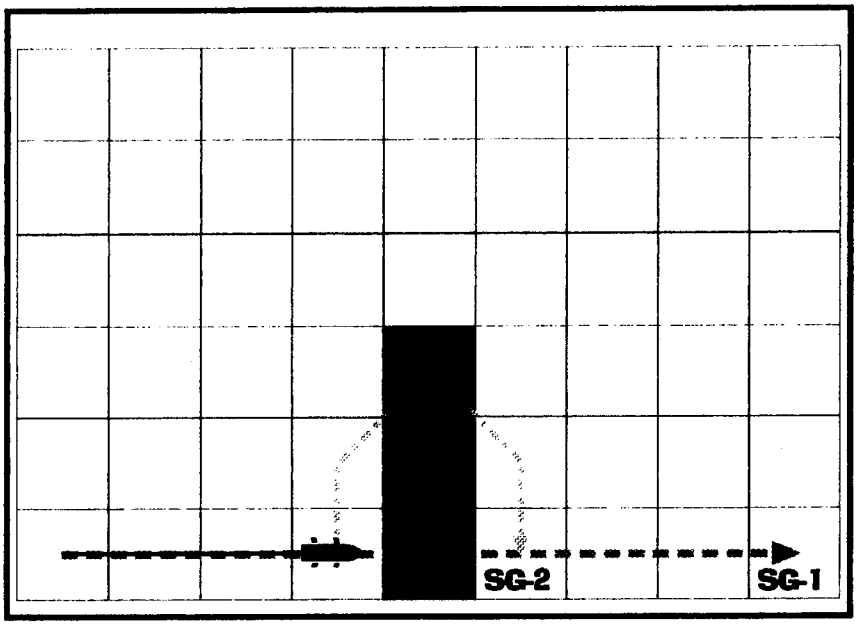

Figure 6: A sub-goal generation with path replanning

During the search, whenever a cell is found unsafe, whether or not on the path, the agenda is checked to verify if this cell is a sub-goal. If so, the sub-goal is taken from the agenda and forgotten. At the same time, a rendering algorithm is called to build a polygon with the unsafe cells found between the AUV path and the sub-goals. If a closed polygon can be formed, the sub-goal is considered unreachable as are all the other sub-goals in the agenda that are within the region defined by this polygon.

Whenever the agenda is found empty, the AUV has reached one of the area limits. It then changes the level according to the value of z-motion and its present status. The next level will be the one that is separated from the present level by a distance equal to two detection ranges. When ready to start the new level,

the x-motion is reset and the whole process begins again. If the agenda is made empty by means of a closed polygon, a new sub-goal will be created at the next level, one cell after the polygon. After sweeping a slice the AUV proceeds to next-slice, starting the whole process again. For the changes of level and slice the path is planned using a hill-climbing procedure. The hill-climbing procedure is a depth-first search modified to sort the choices using the estimated distance to the goal [Ref. 17]. The path is replanned every time a new obstacle is found on the path. After all slices are searched, the algorithm calculates the final fraction of area searched.

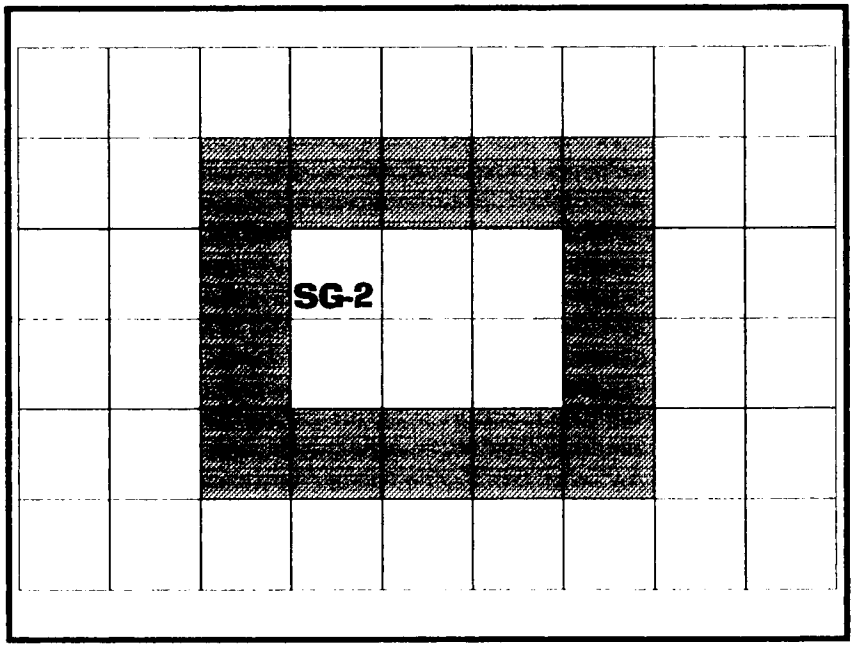


Figure 7: An obstacle containing an unreachable goal

## F. SUMMARY

The world is modeled as a three dimensional matrix where a cell is connected to its 26 neighbors. The obstacle mapping resolution is restricted by the size of each cell. To simplify the algorithm the vehicle has only two degrees of freedom for each maneuver. The proposed algorithm is based on a high level ladder search and the procedure that plans the sweeping path keeps the planned path as close as possible to the ladder pattern. Obstacle avoidance and mapping are performed by establishing sub-goals and pursuing them to define the shape of the obstacle.

# V. EXPERIMENTAL RESULTS

## A. INTRODUCTION

In order to evaluate properly the algorithm and to set a basis for future improvements, different scenarios were used to test the algorithm. The simulation was performed using the Lisp [Ref. 18] code contained in Appendix A. A random mine field generator was also created to allow the use of Monte Carlo simulation techniques. Assessments of the fraction of area searched and total amount of resources needed to perform the missions were made. For the purpose of comparison, an implementation of mine search using the ladder strategy as the high-level behavior and pure hill-climbing for sweeping the levels was developed. The Underwater Virtual World was also used to evaluate and improve the algorithm [Ref. 19]. Figure 8 shows the AUV performing a mission in the Underwater Virtual World.

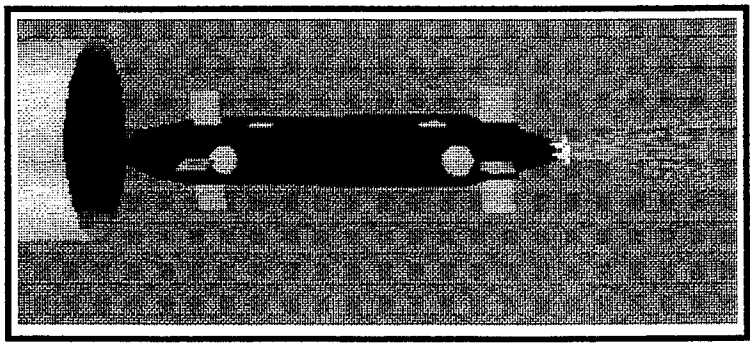

Figure 8: The AUV in the UVW showing an entry into a simulated pipe

**B. FRACTION OF AREA SEARCHED**

The fraction of area searched (FAS) achieved in most of the instances was very high. The average value of 0.78 with an standard deviation of 0.01 was obtained using the most probable scenarios. None of the runs that used from 1 to 50 mines, medium obstacles and mountains, produced a FAS smaller than 0.70. The FAS was substantially affected only when the AUV had to face very large obstacles. The main reason for that reduction was the absence of the rendering procedure as described in the previous chapter. Although the present implementation of the algorithm avoids infinite loops that could be generated by allocating a goal inside a large obstacle, through the use of a subgoal elimination techinique, this occurrence usually reduces the FAS since the cells inside the obstacle continue to be considered unknown cells. Figure 9 shows a large obstacle with the sub-goals caused by it. Sub-goal number two (SG-2) is the first sub-goal generated due to the obstacle. When the AUV returns to the same level as the SG-2, it pops out SG-2, and all other goals above it, from the agenda. SG-2 and the cell to its right were not inspected by the sonar and will remain as unknown cells.

Another reason to have a reduction on the FAS is the proximity to the searched volume boundaries. Whenever the AUV is less than the leg spacing distance from a boundary it considers

the slice as searched and proceeds to the next slice. This was an implementation choice and can be solved in three different ways — by dividing the volume in a number of cells multiple of the leg spacing in each dimension, by adjusting the leg spacing or by forcing a sweep of the volume edge whenever necessary.

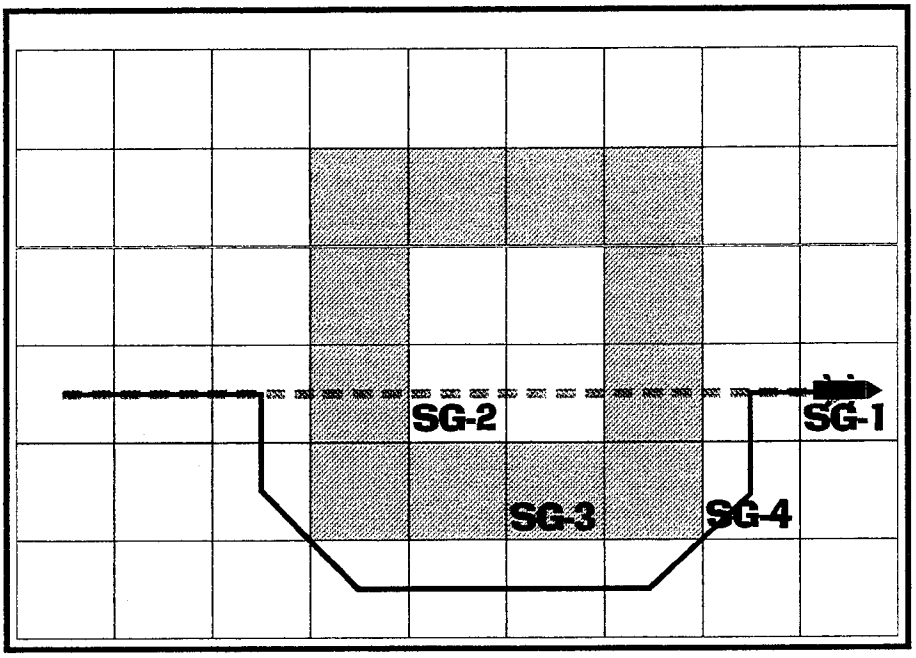


Figure 9: An obstacle and the sub-goals (SG-x) generated to avoid it

Figure 10 displays a graph of the FAS against the number of mines for the sweeping heuristic and the hill-climbing procedure. This graph is a result of 1500 replications of each algorithm. As shown in the graph, the FAS decreases when the number of mines increased. This decrease is a consequence of the formation of virtual barriers or walls due to the proximity of mines. In this

situation the hill-climbing procedure starts to leave shadows as previously shown in Figure 5. The sweeping heuristic also had better performance in scenarios with mountains and other obstacles.

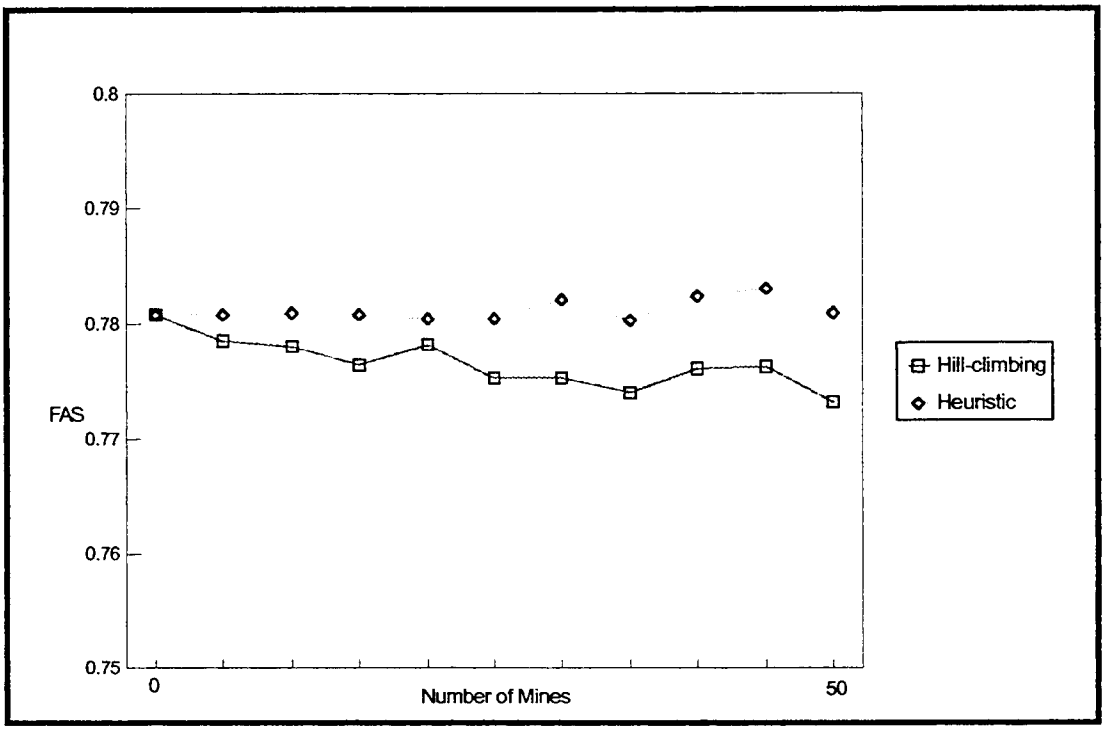


Figure 10: Graph of Number of Mines vs. FAS

Figures 11 and 12 show the behavior of both algorithms in a mine field in a no mountain scenario and in a two mountain scenario respectively.

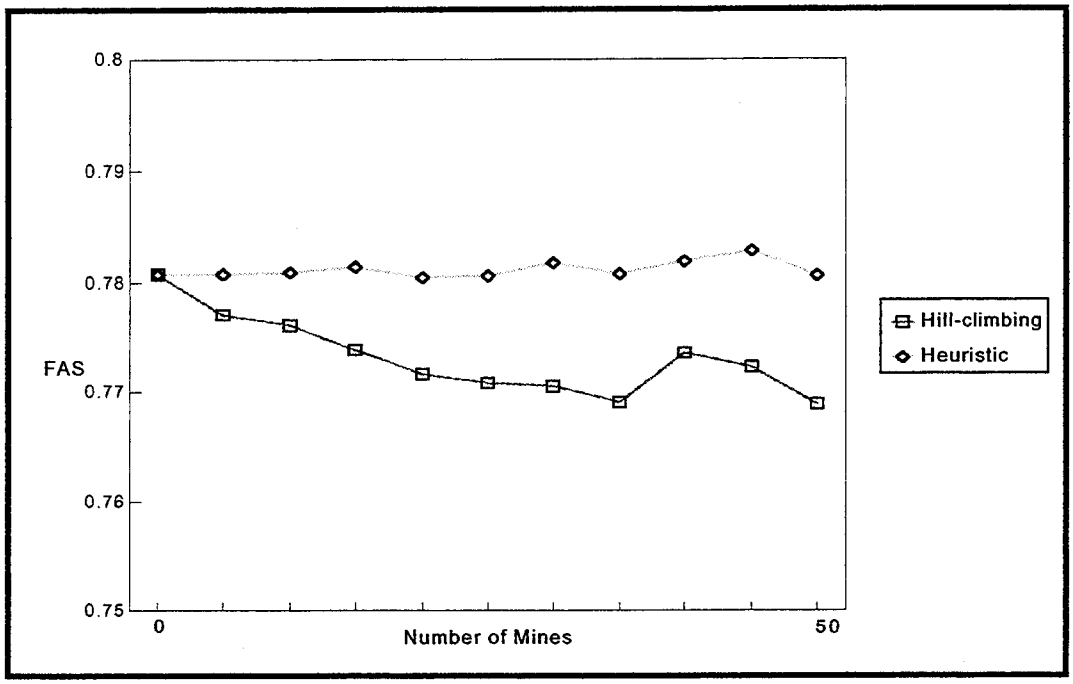


Figure 11: Graph of Number of Mines vs. FAS in a no mountain scenario

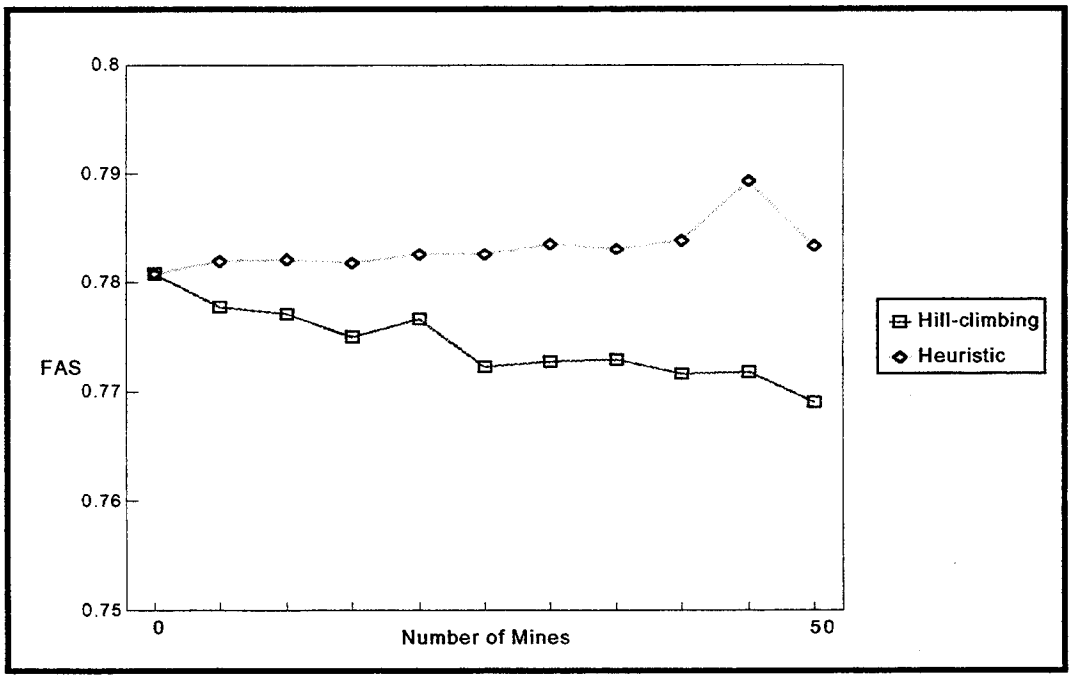


Figure 12: Graph of Number of Mines vs. FAS in a two mountain scenario

## C. RESOURCES UTILIZATION

For each setting, the amount of resources needed to perform the mission was evaluated. The simplest case, with no mines or obstacles, i.e., the AUV needed just to cover the whole volume without obstacle avoidance maneuvers, required 120.0 UR. The worst case tested, where 50 mines were randomly placed in a two mountain scenario, required an average of 123.99 UR with an standard deviation of 7.80 UR. In the same scenario, the hill-climbing procedure required an average of 131.97 with an standard deviation of 15.03 UR. The increase in the number of mines implied an increase in the amount of resources needed, as shown in Figure 13.

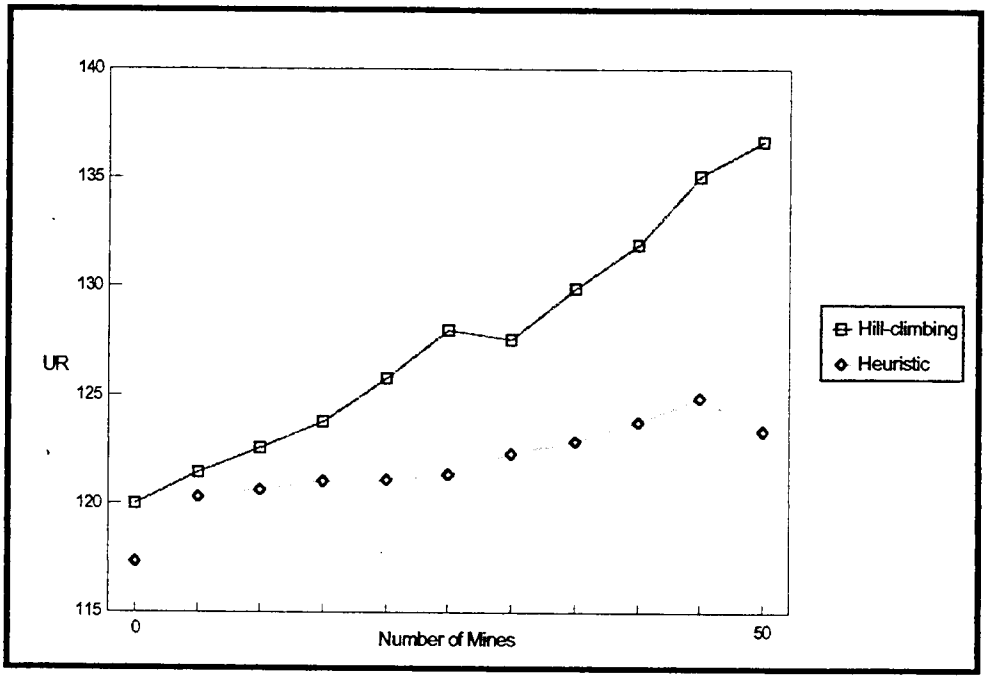


Figure 13: Graph of the Number of Mines vs. UR

The comparison between the sweeping heuristic and the hill-climbing procedure showed that the latter needs more resources to achieve the same results. Figures 14 and 15 show the behavior of both algorithms in a mine field in a no mountain scenario and in a two mountain scenario respectively.

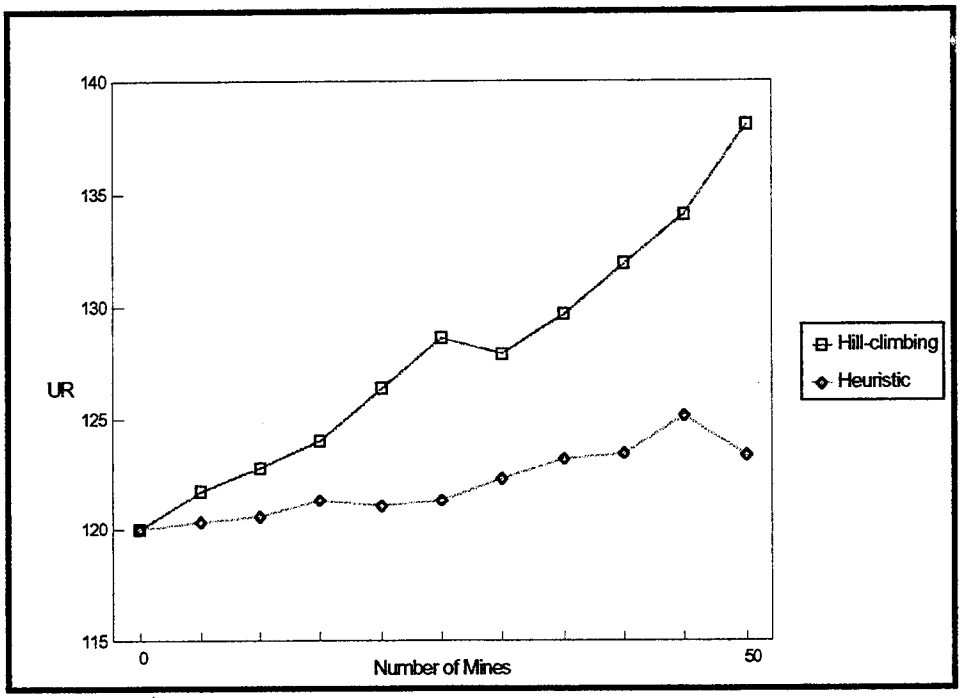


Figure 14: Graph of the Number of Mines vs. UR in a no mountain scenario

Large obstacles did not have a big impact on the amount of resources required. Due to the structure of the algorithm, sometimes large obstacles have a smaller influence on the required

resources than an equivalent number of small obstacles.

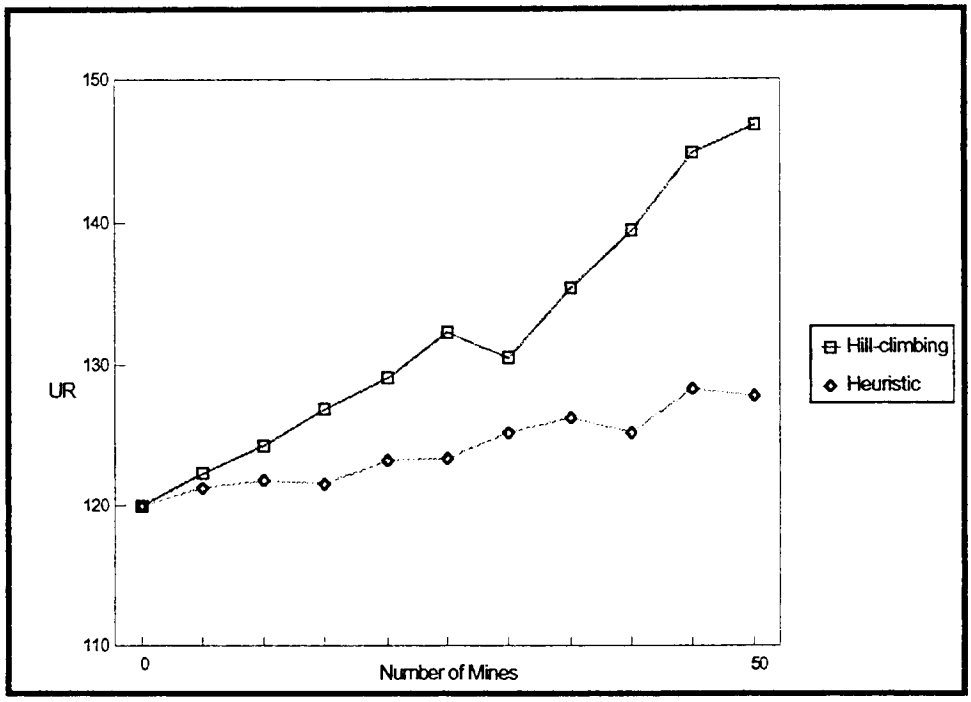


Figure 15: Graph of the Number of Mines vs. UR in a two mountain scenario

**D. SUMMARY**

The simulations of the algorithm were sufficient to gain an overall idea of its performance and behavior. The evaluation of the fraction of area searched and the amount of resources required are important both to judge the algorithm and to establish ground for future developments. The comparison with the performance of a hill-climbing algorithm is important to obtain a fair evaluation of the algorithm.

# VI. RECOMMENDATIONS AND CONCLUSIONS

## A. INTRODUCTION

As was already known, the problem is very hard. The difficulty of searching for mines in an unknown environment allied to the difficulty of guiding a robot made the algorithm complicated.

## B. THE TOOLS

It was certainly very helpful to use Lisp as the language for development. In addition to its natural generality and flexibility that has made Lisp one of the main artificial intelligence languages, the use of an interpreter dramatically facilitates the development and testing. A similar attempt could be made using CLIPS [Ref. 20]. Although CLIPS is not so flexible as Lisp, its rule oriented approach may be a good help in modeling the decision making process.

It is essential that future attempts to develop a new algorithm, or to improve this one, use the Underwater Virtual World (UVW) and the AUV simulator. Besides the robot monitoring capabilities of the simulator, it gives the developer an

interactive visual assessment of the behavior of the robot and therefore the behavior of the algorithm.

### C. THE PROCEDURE

The algorithm that was developed is a first attempt to solve the problem and is certainly not adequate for all possible situations. More cases have to be investigated. The effect of cell size on fraction of the area searched and resources utilization is an important topic that deserves special attention.

The code can probably be improved and the use of other languages and artificial intelligence techniques can open new doors. The use of the profiling sonar can also affect further developments and must be investigated.

### D. THE AUV INTERFACE

To have this algorithm driving the vehicle an interface between the Tactical level, where this algorithm resides, and the Execution level has to be defined. The required communications software has already been developed by the AUV group. It is presently possible to interchange messages between the Tactical and the Execution level, from within the Lisp interpreter. This code is included as Appendix B.

The driving commands have to be issued by the function move-to-waypoint described in Appendix A. The AUV commands described in reference 19 can be used.

The algorithm works with a discrete representation of the world. Therefore, a function is needed to define the correspondence between cells and real world waypoints. Another function is required to perform the sonar search. That function has to send the sonar the appropriate commands, read the related responses and map it to the algorithm representation of the world. This is done by designating each cell that has an echo associated with it an unsafe cell. This function has to replace the look-around function existent in the code.

## E. SUMMARY

As any new technology, the mine search using robots offers many possibilities for research. The development of optimization methods, artificial intelligence techniques, virtual reality and computational resources will provide better solutions to the problem. The evaluation of different tools and techniques has to be used to provide insight for improving the solutions.

The path for linking the algorithm with the real world, i.e., the AUV, is defined. Using the available tools it is possible to use the simulator to perform realistic evaluations of the algorithm behavior without risking the vehicle.

## LIST OF REFERENCES


1.Operations Evaluation Group, Report No. 56, *A Theoretical Basis for Methods of Search and Screening,* by Koopman,B.O., p. vii, Alexandria, Virginia, 1946.

2.Compton,M.A., *Minefield Search and Object Recognition for Autonomous Underwater Vehicles,* Master's Thesis, Naval Postgraduate School, Monterey, California, March 1992.

3.Brutzman,D.P., "From virtual world to reality: designing an autonomous underwater robot," paper presented at American Association for Artificial Intelligence Fall Symposium, Cambridge, Massachussetts, 23 October 1992.

4.Horne III,C.F., "Mine Warfare Is With Us And Will Be With Us," *United States Naval Institute Proceedings,* p. 63, July 1991.

5.Eagle,J.N., Lecture Notes, *Search and Detection Course,* Naval Postgraduate School, January 1994.

6.Hyland,J.C., and Taylor,J., "Mine Avoidance Techniques for Underwater Vehicles," *IEEE Journal of Oceanic Engineering,* v. 18, No. 3, pp. 340-350, July 1993.

7.Byrnes,R.B., Nelson,M.L., Kwak,S.H., McGhee,R.B., Healey,A.J., "Rational Behavior Model: An Implemented Tri-Level Multilingual Software Architecture for Control of Autonomous Underwater Vehicles," *Proceedings of the International Symposium in Unmanned Untethered*

*Submersible Technology*, Durham, New Hampshire, 27 September 1993.

8.Stone,L.D., *Theory of Optimal Search*, Academic Press, New York, 1975.

9.Chudnovsky,D.V., and Chudnovsky,G.V., *Search Theory -- Some Recent Developments*, Marcel Drekker, Inc., New York, New York, 1989.

10.Weisinger,J.R., Monticino,M.G., and Benkoski,S.J., "A Survey of the Search Theory Literature," *Naval Research Logistics*, v.38, pp. 469-494, 1991.

11.Washburn,A.R., *Search and Detection*, 2d ed., pp. 8-1 to 8-13, Operations Research Society of America Books, Arlington, Virginia, 1989.

12.Lumelsky,V.J., Mukhopadhyay,S., and Sun,K., "Dynamic Path Planning in Sensor-Based Terrain Acquisition," *IEEE Transactions on Robotics and Automation*, v. 6, no. 4, pp. 530-540, 1990.

13.Krogh,B.H., and Feng,D., "Dynamic Generation of Subgoals for Autonomous Mobile Robots Using Local Feedback Information," *IEEE Transactions on Automatic Control*, v. 34, no. 5, pp. 465-475, 1989.

14.Schwartz,J.T., and Sharir,M., "A Survey of Motion Planning and Related Geometric Algorithms," *Artificial Intelligence Journal*, v. 37, pp. 157-169, 1988.

15.Caddell,T.W., *Three-Dimensional Path Planning for the NPS II AUV*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1991.

16.Durrant-Whyte,H.F., Leonard,J.J., and Cox,I.J., "Dynamic Map Building for an Autonomous Mobile Robot," *Proceedings of the IEEE International Workshop on*

*Intelligent Robots and Systems (IROS 90)*, pp. 89-95, 1990.

17.Winston,P.H., *Artificial Intelligence*, 3rd ed., p. 70, Addison-Wesley Publishing Co., Reading, Massachusets, 1992.

18.Steele,G.L., *Common Lisp -- The Language*, 2d ed., Digital Press, Lexington, Massachusetts, 1990.

19.Brutzman,D.P., *A Virtual World for an Autonomous Underwater Vehicle*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, December 1994.

20.Giarratano,J.C., *CLIPS User's Guide*, version 6.0, NASA, Lyndon B. Johnson Space Center, May 1993.

## APPENDIX A. THE MINE SEARCH SOURCE CODE

This Appendix contains the Lisp code that implements the algorithm developed in this thesis.

```
;;;
```

```
(defstruct auv                ;; structure to define an auv and its location
    (x-position 0 :type integer)
    (y-position 0 :type integer)
    (z-position 0 :type integer)
    (status 'sweeping-level :type auv-status))

(defvar *phoenix* (make-auv))  ;; our auv

(deftype cell-state () '(member unknown safe visited unsafe)) ;; possible states of a cell

(defvar *area*
    (make-array '(11 11 11)
        :element-type 'cell-state
        :initial-element 'safe))

(defvar *auv-map*
    (make-array '(11 11 11)
        :element-type 'cell-state
        :initial-element 'unknown))
```

```
; Implementation of AUVMINE in Lisp

; Author: Jose A. Rodrigues Nt.
; Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
; Date: September 1994
; Version: 0.5
; Compiler: Franz Allegro CL/PC version 1.0
;
; Function init-search: Initializes all the variables needed for the algorithm.
; Only called once.
;
; Function restart-search: Reinitialize some of the variables at each sweep-area call.
;
; Function define-area: Set all the cells beyond the area limits as obstacles.
; Only called for the whole area.
;
; Function init-agenda: Prepare the agenda, initializing it with the first sub-goal for
; the present search.
;
; Function set-level-goal: Defines the the sub-goal at the end of the level.


(defun init-search (initial-waypoint final-waypoint)
    (define-area)
    (setf (auv-x-position *phoenix*) (car initial-waypoint))
    (setf (auv-y-position *phoenix*) (cadr initial-waypoint))
    (setf (auv-z-position *phoenix*) (caddr initial-waypoint))
    (setf *x-motion* (/ (- (car final-waypoint) (car initial-waypoint))
                                    (abs (- (car final-waypoint) (car initial-waypoint)))))
    (setf *y-motion* (/ (- (cadr final-waypoint) (cadr initial-waypoint))
                                (abs (- (cadr final-waypoint) (cadr initial-waypoint)))))
    (setf *z-motion* (/ (- (caddr final-waypoint) (caddr initial-waypoint))
                            (abs (- (caddr final-waypoint) (caddr initial-waypoint)))))
    (setf (auv-status *phoenix*) 'sweeping-level)
    (setf *sweep-path* '())
    (setf *used-path* '())
    (init-agenda initial-waypoint final-waypoint))
```

```
(defun restart-search (initial-waypoint final-waypoint)
;; needed to restart a search when it's called for a subarea.
;; Init-search can not be used since the AUV will already be in position
;; and *used-path* contains the previous search log.
    (setf *x-motion* (/ (- (car final-waypoint) (car initial-waypoint))
                                        (abs (- (car final-waypoint) (car initial-waypoint)))))
    (setf (auv-status *phoenix*) 'sweeping-level)
    (setf *sweep-path* '())
    (init-agenda initial-waypoint final-waypoint))

(defun define-area ()
;; set all nodes in the boundary as obstacles
;; it is defined "hardwired" now but could use the car and cdr
;; from initial and final waypoints
    (loop
        with x = 0
        and  y = 0
        and  z = 0
        for x in '(0 10)
            do (loop for  y from 0 to 10
                do (loop for z  from 0 to 10
                    do (setf (aref *auv-map* x y z) 'unsafe)
                        (setf (aref *area* x y z) 'unsafe))))
    (loop
        with  x = 0
        and  y =  0
        and  z = 0
        for  y in '(0 10)
            do (loop for  x from 0 to 10
                do (loop for z from 0 to 10
                    do (setf (aref *auv-map* x y z) 'unsafe)
                        (setf (aref *area* x y z) 'unsafe))))
    (loop
        with  x = 0
        and  y = 0
        and  z = 0
        for z in '(0 10)
            do (loop for y from 0 to 10
                do (loop for x from 0 to 10
                    do (setf (aref *auv-map* x y z) 'unsafe)
                        (setf (aref *area* x y z) 'unsafe)))))
```

```
(defun init-agenda (initial-waypoint final-waypoint)
;; defines the first level goal and inserts it into the agenda
    (setf *agenda* nil)
    (push (set-level-goal initial-waypoint final-waypoint) *agenda*))

(defun set-level-goal (initial-waypoint final-waypoint)
    (setf *level-goal* (cons (car final-waypoint) (cons (cadr initial-waypoint)
                                                        (cons (caddr initial-waypoint) nil)))))
```

```
; Implementation of AUVMINE in Lisp

; Author: Jose A. Rodrigues Nt.
; Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
; Date: September 1994
; Version: 0.5
; Compiler: Franz Allegro CL/PC version 1.0

(setf *default-pathname-defaults* #P"D:\\ALLEGRO\\AUVMINE\\")

(load "auv_var.cl") ;; loads the variables' definitions

(load "auv_look.cl") ;; loads the look-around function

(load "auv_chec.cl") ;; loads the check-plane function

(load "auv_init.cl")  ;; loads the initializaton functions

(load "auv_splt.cl")  ;; loads the split-area function

(load "auv_tran.cl")  ;; loads the transit function

(load "auv_larg.cl")  ;; loads the large-obstacle function

(load "auv_wall.cl")  ;; loads the check-wall function

(load "auv_hill.cl") ;; loads the hill-path function

(load "auv_hil2.cl") ;; loads the hill-path2 function

(load "auv_prob.cl")  ;; loads the calc-prob function

(load "auv_sim.cl")  ;; loads the functions that produce data to be analysed at the UVW

(load "auv_cost.cl")  ;; loads the cost evaluation function


(setf *default-pathname-defaults* #P"D:\\ALLEGRO\\")

;; legal-movep has also to check if location is inside a closed polygon. Not  implemented.
(defun legal-movep (x-location y-location z-location)
;; checks the map for a prospect move. avoids circle.
    (if (not (equal (aref *auv-map* x-location y-location z-location) 'unsafe)) t))
```

```
(defun find-path (sub-goal)
;; find the best path from the current position to the present sub-goal
   (do* ((planned-step (list (auv-x-position *phoenix*) (auv-y-position *phoenix*)
         (auv-z-position *phoenix*))
      (cond ((plusp (/ (- (car planned-step) (car sub-goal)) *x-motion*))
                (pop *agenda*)
                (setq sub-goal (car *agenda*))
                (setq *sweep-path* nil)
                (setq planned-step (list (auv-x-position *phoenix*)
                                         (auv-y-position *phoenix*)
                                         (auv-z-position *phoenix*)))
                (car (update-sweep-path (best-move planned-step))))
            (t (car (update-sweep-path (best-move planned-step)))))))
      ((equal planned-step sub-goal) (setf *sweep-path* (reverse *sweep-path*)))))

(defun plan-path (sub-goal)
   (cond
      ((equal (auv-status *phoenix*) 'changing-level)
         (hill-path (list (auv-x-position *phoenix*) (auv-y-position *phoenix*)
                          (auv-z-position *phoenix*)) sub-goal))
      ((equal (auv-status *phoenix*) 'changing-slice)
         (hill-path2 (list (auv-x-position *phoenix*) (auv-y-position *phoenix*)
                          (auv-z-position *phoenix*)) sub-goal))
      (t (find-path sub-goal))))
```

```
(defun best-move (position)
;; define the best move from position according to our rules.
    (case (auv-status *phoenix*)
        ('sweeping-level
            (if (equal (caddr position) (caddr *level-goal*))
                (cond
                    ((legal-movep (+ (car position) *x-motion*)  (cadr position)
                                                        (caddr position))
                        (setf position (cons (+ (car position) *x-motion*) (cdr position))))
                    ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                        (+ (caddr position) *z-motion*))
                        (setf position (cons (+ (car position) *x-motion*)
                                (cons (cadr position)  (cons (+ (caddr position) *z-motion*)
                                                                        nil)))))
                    ((legal-movep (car position) (cadr position)
                                                        (+ (caddr position) *z-motion*))
                        (setf position (cons (car position) (cons (cadr position)
                                        (cons (+ (caddr position) *z-motion*) nil)))))
                    ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                        (+ (caddr position) *z-motion*))
                        (setf position (cons (- (car position) *x-motion*)
                            (cons (cadr position) (cons (+ (caddr position) *z-motion*)
                                                                        nil)))))
                    ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                                (caddr position))
                        (setf position (cons (- (car position) *x-motion*) (cdr position))))
                    ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                        (- (caddr position) *z-motion*) )
                        (setf position (cons (+ (car position) *x-motion*)
                                (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                                        nil)))))
                    ((legal-movep (car position) (cadr position)
                                                        (- (caddr position) *z-motion*))
                        (setf position (cons (car position) (cons (cadr position)
                                                (cons (- (caddr position) *z-motion*) nil)))))
                    ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                        (- (caddr position) *z-motion*))
                        (setf position (cons (- (car position) *x-motion*)
                                (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                                        nil)))))
                    (t print "wrong find-path"))
```

```
(cond
   ((and (legal-movep (car position) (cadr position)
                                        (- (caddr position) *z-motion*))
   (not (equal (aref *auv-map* (car position) (cadr position)
                                (- (caddr position) *z-motion*)) ' visited)))
      (setf position (cons (car position) (cons (cadr position)
                                (cons (- (caddr position) *z-motion*) nil)))))
   ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                        (- (caddr position) *z-motion*))
      (setf position (cons (+ (car position) *x-motion*)
         (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                              nil)))))
   ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                    (caddr position))
      (setf position (cons (+ (car position) *x-motion*) (cdr position))))
   ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                        (+ (caddr position) *z-motion*))
      (setf position (cons (+ (car position) *x-motion*)
         (cons (cadr position) (cons (+ (caddr position) *z-motion*)
                                                              nil)))))
   ((legal-movep (car position) (cadr position)
                                        (+ (caddr position) *z-motion*))
      (setf position (cons (car position) (cons (cadr position)
                           (cons (+ (caddr position) *z-motion*) nil)))))
   ((legal-movep (- (car position) *x-motion*) (cadr position)
                                        (+ (caddr position) *z-motion*))
      (setf position (cons (- (car position) *x-motion*)
         (cons (cadr position) (cons (+ (caddr position) *z-motion*)
                                                              nil)))))
   ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                    (caddr position))
      (setf position (cons (- (car position) *x-motion*) (cdr position))))
   ((legal-movep (- (car position) *x-motion*) (cadr position)
                                        (- (caddr position) *z-motion*))
      (setf position (cons (- (car position) *x-motion*)
         (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                              nil)))))
   (t print "wrong find-path"))))
```

```
('changing-level
   (if (equal (car position) (car *level-goal*))
      (cond
         ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                (+ (caddr position) *z-motion*))
            (setf position (cons (+ (car position) *x-motion*)
               (cons (cadr position) (cons (+ (caddr position) *z-motion*)
                                                                  nil)))))
         ((legal-movep (car position) (cadr position)
                                                (+ (caddr position) *z-motion*))
            (setf position (cons (car position) (cons (cadr position)
                              (cons (+ (caddr position) *z-motion*) nil)))))
         ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                (+ (caddr position) *z-motion*))
            (setf position (cons (- (car position) *x-motion*)
               (cons (cadr position) (cons (+ (caddr position) *z-motion*)
                                                                  nil)))))
         ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                            (caddr position))
            (setf position (cons (- (car position) *x-motion*) (cdr position))))
         ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                (- (caddr position) *z-motion*))
            (setf position (cons (- (car position) *x-motion*)
               (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                                  nil)))))
         ((legal-movep (car position) (cadr position)
                                                (- (caddr position) *z-motion*))
            (setf position (cons (car position) (cons (cadr position)
                                 (cons (- (caddr position) *z-motion*) nil)))))
         ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                (- (caddr position) *z-motion*) )
            (setf position (cons (+ (car position) *x-motion*)
               (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                                  nil)))))
         ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                            (caddr position))
            (setf position (cons (+ (car position) *x-motion*) (cdr position))))
         (t print "wrong find-path"))
```

```
(cond
   ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                    (caddr position))
      (setf position (cons (+ (car position) *x-motion*) (cdr position))))
   ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                          (+ (caddr position) *z-motion*))
      (setf position (cons (+ (car position) *x-motion*)
         (cons (cadr position) (cons (+ (caddr position) *z-motion*)
                                                              nil)))))
   ((legal-movep (car position) (cadr position)
                                          (+ (caddr position) *z-motion*))
      (setf position (cons (car position) (cons (cadr position)
                        (cons (+ (caddr position) *z-motion*) nil)))))
   ((legal-movep (- (car position) *x-motion*) (cadr position)
                                          (+ (caddr position) *z-motion*))
      (setf position (cons (- (car position) *x-motion*)
         (cons (cadr position) (cons (+ (caddr position) *z-motion*)
                                                              nil)))))
   ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                    (caddr position))
      (setf position (cons (- (car position) *x-motion*) (cdr position))))
   ((legal-movep (- (car position) *x-motion*) (cadr position)
                                          (- (caddr position) *z-motion*))
      (setf position (cons (- (car position) *x-motion*)
         (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                              nil)))))
   ((legal-movep (car position) (cadr position)
                                          (- (caddr position) *z-motion*))
      (setf position (cons (car position) (cons (cadr position)
                              (cons (- (caddr position) *z-motion*) nil)))))
   ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                          (- (caddr position) *z-motion*) )
      (setf position (cons (+ (car position) *x-motion*)
         (cons (cadr position) (cons (- (caddr position) *z-motion*)
                                                              nil)))))
   (t print "wrong find-path"))))
```

```
('changing-slice
   (if (equal (car position) (car *level-goal*))
      (cond
         ((legal-movep (+ (car position) *x-motion*)
                           (+ (cadr position) *y-motion*) (caddr position))
            (setf position (cons (+ (car position) *x-motion*)
               (cons (+ (cadr position) *y-motion*) (cons (caddr position)
                                                   nil)))))
         ((legal-movep (car position) (+ (cadr position) *y-motion*)
                                                   (caddr position))
            (setf position (cons (car position)
               (cons (+ (cadr position) *y-motion*) (cons (caddr position)
                                                   nil)))))
         ((legal-movep (- (car position) *x-motion*)
                           (+ (cadr position) *y-motion*) (caddr position))
            (setf position (cons (- (car position) *x-motion*)
               (cons (+ (cadr position) *y-motion*) (cons (caddr position)
                                                   nil)))))
         ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                   (caddr position))
            (setf position (cons (- (car position) *x-motion*)
                     (cons (cadr position) (cons (caddr position) nil)))))
         ((legal-movep (- (car position) *x-motion*)
                           (- (cadr position) *y-motion*) (caddr position))
            (setf position (cons (- (car position) *x-motion*)
               (cons (- (cadr position) *y-motion*) (cons (caddr position)
                                                   nil)))))
         ((legal-movep (car position) (- (cadr position) *y-motion*)
                                                   (caddr position))
            (setf position (cons (car position)
               (cons (- (cadr position) *y-motion*) (cons (caddr position)
                                                   nil)))))
         ((legal-movep (+ (car position) *x-motion*)
               (- (cadr position) *y-motion*) (- (caddr position) *z-motion*) )
            (setf position (cons (+ (car position) *x-motion*)
               (cons (- (cadr position) *y-motion*)
                           (cons (- (caddr position) *z-motion*) nil)))))
         ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                   (caddr position))
            (setf position (cons (+ (car position) *x-motion*) (cdr position))))
         (t print "wrong find-path"))
```

```
(cond
   ((legal-movep (+ (car position) *x-motion*) (cadr position)
                                                    (caddr position))
      (setf position (cons (+ (car position) *x-motion*) (cdr position))))
   ((legal-movep (+ (car position) *x-motion*)
                        (+ (cadr position) *y-motion*) (caddr position))
      (setf position (cons (+ (car position) *x-motion*)
         (cons (+ (cadr position) *y-motion*) (cons (caddr position)
                                                          nil)))))
   ((legal-movep (car position) (+ (cadr position) *y-motion*)
                                                    (caddr position))
      (setf position (cons (car position)
         (cons (+ (cadr position) *y-motion*) (cons (caddr position)
                                                          nil)))))
   ((legal-movep (- (car position) *x-motion*)
                        (+ (cadr position) *y-motion*) (caddr position))
      (setf position (cons (- (car position) *x-motion*)
         (cons (+ (cadr position) *y-motion*) (cons (caddr position)
                                                          nil)))))
   ((legal-movep (- (car position) *x-motion*) (cadr position)
                                                    (caddr position))
      (setf position (cons (- (car position) *x-motion*)
                        (cons (cadr position) (cons (caddr position) nil)))))
   ((legal-movep (- (car position) *x-motion*)
                        (- (cadr position) *y-motion*) (caddr position))
      (setf position (cons (- (car position) *x-motion*)
         (cons (- (cadr position) *y-motion*) (cons (caddr position)
                                                          nil)))))
   ((legal-movep (car position) (- (cadr position) *y-motion*)
                                                    (caddr position))
      (setf position (cons (car position)
         (cons (- (cadr position) *y-motion*) (cons (caddr position)
                                                          nil)))))
   ((legal-movep (+ (car position) *x-motion*)
         (- (cadr position) *y-motion*) (- (caddr position) *z-motion*) )
      (setf position (cons (+ (car position) *x-motion*)
         (cons (- (cadr position) *y-motion*)
                              (cons (- (caddr position) *z-motion*) nil)))))
   (t print "wrong find-path"))))))

(defun update-sweep-path (new-step)
   (setf *sweep-path* (cons new-step *sweep-path*)))
```

```
(defun move-to-waypoint (location)  ;; changes the auv position to the location
    (setf (auv-x-position *phoenix*) (car location))
    (setf (auv-y-position *phoenix*) (cadr location))
    (setf (auv-z-position *phoenix*) (caddr location))
    (setf (aref *auv-map* (car location) (cadr location) (caddr location)) 'visited)
    (setf *used-path* (cons (cons (car location) (cons (cadr location)
                                        (cons (caddr location) nil))) *used-path*)))

(defun next-safe () ;; checks if the next move in the planned path is safe
    (legal-movep (car (car *sweep-path*)) (cadr (car *sweep-path*))
                                                (caddr (car *sweep-path*))))
```

```
(defun sweep-area (initial-waypoint final-waypoint)
    (restart-search initial-waypoint final-waypoint)
    (loop ;; area loop. will get out when whole area is done
        until (equal (auv-status *phoenix*) 'area-done)
        do
            (loop  ;; level loop. when we get out of it we have to change status
                until (null *agenda*)
                do (cond ((and (legal-movep (car (car *agenda*)) (cadr (car *agenda*))
                                                                    (caddr (car *agenda*)))
                               (not(left-back (car *agenda*))))
                          (plan-path (car *agenda*))
                          (loop  ;; do the real work in the level, moving and replanning
                                 ;; when needed
                              until (equal (cons (auv-x-position *phoenix*)
                                                 (cons (auv-y-position *phoenix*)
                                                 (cons (auv-z-position *phoenix*) nil)))
                                           (car *agenda*))
                              do (unless (legal-movep (car (car *agenda*))
                                                 (cadr (car *agenda*)) (caddr (car *agenda*)))
                                     (pop *agenda*)
                                     (plan-path (car *agenda*)))
                          do (cond ((left-back (nth (- (list-length *agenda*) 2) *agenda*))
                                     (large-obstacle)
                                     (plan-path (car *agenda*))))
                              when (next-safe)
                                  do (move-to-waypoint (pop *sweep-path*))
                                                                    ;; move to next waypoint
                                  and do (look-around)  ;; update map
                                  and do (if (check-wall)
                                             (split-area initial-waypoint final-waypoint))
                              else
                                  do (push (new-subgoal) *agenda*)
                                  and do (setf *sweep-path* 'nil)
                                  and do (if (check-wall)
                                             (split-area initial-waypoint final-waypoint)
                                             (plan-path (car *agenda*))))))
                (pop *agenda*))
```

```
        (let*
            ((north-limit (max (car initial-waypoint) (car final-waypoint)))
            (south-limit (min (car initial-waypoint) (car final-waypoint)))
            (up-limit (min (caddr initial-waypoint) (caddr final-waypoint)))
            (down-limit (max (caddr initial-waypoint) (caddr final-waypoint)))
            (east-limit (max (cadr initial-waypoint) (cadr final-waypoint)))
            (west-limit (min (cadr initial-waypoint) (cadr final-waypoint))))
            (change-status north-limit south-limit up-limit down-limit
                                                    east-limit west-limit))))

(defun new-subgoal ()
    (case (auv-status *phoenix*)
        ('sweeping-level (cons (+ (car (car *sweep-path*)) *x-motion*)
                               (cons (cadr (car *sweep-path*))
                               (cons (caddr (car *sweep-path*)) nil))))
        ('changing-level (cons (car (car *sweep-path*)) (cons (cadr (car *sweep-path*))
                                    (cons (+ (caddr (car *sweep-path*)) *z-motion*) nil))))
        ('changing-slice (cons (car (car *sweep-path*))
                                        (cons (+ (cadr (car *sweep-path*)) *y-motion*)
                                        (cons (caddr (car *sweep-path*)) nil))))))

(defun change-status (north-limit south-limit up-limit down-limit east-limit west-limit)
;; analyses the auv status and location and change it accordingly
    (case (auv-status *phoenix*)
        ('sweeping-level
            (cond
                ((and (>= (+ (caddr *level-goal*) (* 3 *z-motion*)) up-limit)
                        (<= (+ (caddr *level-goal*) (* 3 *z-motion*)) down-limit))
                    (push (cons (car *level-goal*) (cons (cadr *level-goal*)
                        (cons (+ (caddr *level-goal*) (* 3 *z-motion*)) nil))) *agenda*)
                    (setf (auv-status *phoenix*) 'changing-level))
                ((and (or (< (+ (caddr *level-goal*) (* 3 *z-motion*)) up-limit)
                            (> (+ (caddr *level-goal*) (* 3 *z-motion*)) down-limit))
                        (and (>= (+ (cadr *level-goal*) (* 3 *y-motion*)) west-limit)
                            (<= (+ (cadr *level-goal*) (* 3 *y-motion*)) east-limit)))
                    (push (cons (car *level-goal*) (cons (+ (cadr *level-goal*)
                            (* 3 *y-motion*)) (cons (caddr *level-goal*) nil))) *agenda*)
                    (setf (auv-status *phoenix*) 'changing-slice))
                (t (setf (auv-status *phoenix*) 'area-done))))
```

```
        ('changing-level
            (if (plusp *x-motion*)
                (setf *level-goal* (cons south-limit (cons (cadr *level-goal*)
                                    (cons (+ (caddr *level-goal*) (* 3 *z-motion*)) nil))))
                (setf *level-goal* (cons north-limit (cons (cadr *level-goal*)
                                    (cons (+ (caddr *level-goal*) (* 3 *z-motion*)) nil)))))
            (push *level-goal* *agenda*)
            (setf *x-motion* (* -1 *x-motion*))
            (setf (auv-status *phoenix*) 'sweeping-level))
        ('changing-slice
            (if (plusp *x-motion*)
                (setf *level-goal* (cons south-limit (cons (+ (cadr *level-goal*)
                                        (* 3 *y-motion*)) (cons (caddr *level-goal*) nil))))
                (setf *level-goal* (cons north-limit (cons (+ (cadr *level-goal*)
                                        (* 3 *y-motion*)) (cons (caddr *level-goal*) nil)))))
            (push *level-goal* *agenda*)
            (setf *x-motion* (* -1 *x-motion*))
            (setf *z-motion* (* -1 *z-motion*))
            (setf (auv-status *phoenix*) 'sweeping-level))))

(defun search-mine (initial-waypoint final-waypoint)
    (init-search initial-waypoint final-waypoint)
    (sweep-area initial-waypoint final-waypoint)
    (pprint (calc-prob *auv-map*))
    (build-sim-path (reverse *used-path*))
```

```
; Implementation of a Hillclimb procedure for the AUV mine search algorithm

;  Author: Jose A. Rodrigues Nt.
;  Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
;  Date: September 1994
;  Version: 0.5
;  Compiler: Franz Allegro CL/PC version 1.0
;
; Remarks: This implementation is an adaptation of the Hillclimb-array code from
;              the class CS4314 Symbolic Computing conducted by Dr. Robert McGhee.


(defun obstaclep (location) ; location is *auv-map* index list '(row column)
    (if (equal (aref *auv-map* (first location) (second location) (third location)) 'unsafe)
                                                                                t))

(defun legal-move-list (location)
    (remove nil (list (up-movep location) (down-movep location)
        (north-movep location) (south-movep location))))

(defun hill-legal-movep (location)
    (if (and (not (obstaclep location))
             (not (member location *virtual-obstacle-list* :test #'equal))
             (not (member location *sweep-path* :test #'equal))) location))

(defun north-movep (location)
    (hill-legal-movep (cons (1+ (first location)) (rest location))))

(defun south-movep (location)
    (hill-legal-movep (cons (1- (first location)) (rest location))))

(defun up-movep (location)
    (hill-legal-movep (list (first location) (second location) (1- (third location)))))

(defun down-movep (location)
    (hill-legal-movep (list (first location) (second location) (1+ (third location)))))

(defun evalsort (list) (sort list #'closer-to-goalp))

(defun distance-to-goal (location)
    (let ((deltax (- (first location) (first *goal*)))
        (deltaz (- (third location) (third *goal*))))
        (sqrt(+ (* deltax deltax) (* deltaz deltaz)))))
```

```
(defun closer-to-goalp (location1 location2) ;Uses global variable *hill-goal*.
    (if (< (distance-to-goal location1) (distance-to-goal location2)) t))
(defun move-from (location)
    (cond ((null (legal-move-list location)) (backtrack-from location))
    (t (make-best-move-from location))))

(defun make-best-move-from (location)
    (let ((best-move (first (evalsort (legal-move-list location)))))
    (setf *sweep-path* (cons location *sweep-path*)) best-move))

(defun backtrack-from (location)
    (push location *virtual-obstacle-list*)
    (pop *sweep-path*))

(defun hill-path (start goal)
    (setf *sweep-path* nil)
    (setf *virtual-obstacle-list* nil)
    (setf *goal* goal)
    (do ((location start (move-from location)))
        ((or (equal location goal) (null location))
        (if (equal location goal) (push location *sweep-path*))))
    (setf *sweep-path* (reverse (butlast *sweep-path*))))
```

```
; Implementation of a Hillclimb procedure for the AUV mine search algorithm
;  Author: Jose A. Rodrigues Nt.
;  Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
;  Date: September 1994
;  Version: 0.5
;  Compiler: Franz Allegro CL/PC version 1.0
;
; Remarks: This implementation is an adaptation of the Hillclimb-array code from
;          the class CS4314 Symbolic Computing conducted by Dr. Robert McGhee.
;
; Function hill-path2: uses a hill-climbing procedure to find the shortest path between
;                      two points located in different slices. Maneuvers only in X and Y.


(defun legal-move-list2 (location)
   (remove nil (list (east-movep location) (west-movep location)
                     (north-movep location) (south-movep location))))

(defun east-movep (location)
   (hill-legal-movep (list (first location) (1+ (second location)) (third location))))

(defun west-movep (location)
   (hill-legal-movep (list (first location) (1- (second location)) (third location))))

(defun distance-to-goal2 (location)
   (let ((deltax (- (first location) (first *goal*)))
      (deltay (- (second location) (second *goal*))))
      (sqrt(+ (* deltax deltax) (* deltay deltay)))))

(defun closer-to-goal2p (location1 location2) ;Uses global variable *hill-goal*.
   (if (< (distance-to-goal2 location1) (distance-to-goal2 location2)) t))

(defun move-from2 (location)
   (cond
      ((null (legal-move-list2 location)) (backtrack-from location))
      (t (make-best-move-from2 location))))

(defun make-best-move-from2 (location)
   (let
      ((best-move (first (evalsort (legal-move-list2 location)))))
      (setf *sweep-path* (cons location *sweep-path*)) best-move))
```

```
(defun hill-path2 (start goal)
    (setf *sweep-path* nil)
    (setf *virtual-obstacle-list* nil)
    (setf *goal* goal)
    (do ((location start (move-from2 location)))
        ((or (equal location goal) (null location))
        (if (equal location goal) (push location *sweep-path*))))
    (setf *sweep-path* (reverse (butlast *sweep-path*))))
```

```
;  Implementation of AUVMINE in Lisp

;  Author: Jose A. Rodrigues Nt.
;  Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
;  Date: September 1994
;  Version: 0.5
;  Compiler: Franz Allegro CL/PC version 1.0
;
;  Function look-around:  updates the auv map based on the area map


(defun look-around () ;;
    (case (auv-z-position *phoenix*)
        (0 (check-plane (auv-z-position *phoenix*))
                                    (check-plane (+ (auv-z-position *phoenix*) 1)))
        (8 (check-plane (auv-z-position *phoenix*))
                                    (check-plane (- (auv-z-position *phoenix*) 1)))
        (otherwise (check-plane (auv-z-position *phoenix*))
                                    (check-plane (+ (auv-z-position *phoenix*) 1))
                                    (check-plane (- (auv-z-position *phoenix*) 1)))))
```

```
; Implementation of AUVMINE in Lisp

; Author: Jose A. Rodrigues Nt.
; Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
; Date: September 1994
; Version: 0.5
; Compiler: Franz Allegro CL/PC version 1.0
;
; Function check-plane: Copies the 9 cells centered at the AUV x and y position at
; the corresponding level, from the area-map to the auv-map. This is the function that
; emulates the sonar. The area-map is the representation of the area and is what is seen
; by the AUV sonar.

(defun check-plane (level)
    (case (auv-x-position *phoenix*)
        (0
            (case (auv-y-position *phoenix*)
                (0
                    (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                (auv-y-position *phoenix*) level)
                        (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                (auv-y-position *phoenix*) level))
                    (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                (+ (auv-y-position *phoenix*) 1) level)
                        (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                (+ (auv-y-position *phoenix*) 1) level))
                    (setf (aref *auv-map* (auv-x-position *phoenix*)
                                                (+ (auv-y-position *phoenix*) 1) level)
                        (aref *area* (auv-x-position *phoenix*)
                                                (+ (auv-y-position *phoenix*) 1) level)))
                (8
                    (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                    (auv-y-position *phoenix*) level)
                        (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                (auv-y-position *phoenix*) level))
                    (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                (- (auv-y-position *phoenix*) 1) level)
                        (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                (- (auv-y-position *phoenix*) 1) level))
                    (setf (aref *auv-map* (auv-x-position *phoenix*)
                                                (- (auv-y-position *phoenix*) 1) level)
                        (aref *area* (auv-x-position *phoenix*)
                                                (- (auv-y-position *phoenix*) 1) level)))
```

```
            (otherwise
                (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                         (auv-y-position *phoenix*) level)
                    (aref *area* (+ (auv-x-position *phoenix*) 1)
                                         (auv-y-position *phoenix*) level))
                (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                         (+ (auv-y-position *phoenix*) 1) level)
                    (aref *area* (+ (auv-x-position *phoenix*) 1)
                                         (+ (auv-y-position *phoenix*) 1) level))
                (setf (aref *auv-map* (auv-x-position *phoenix*)
                                         (+ (auv-y-position *phoenix*) 1) level)
                    (aref *area* (auv-x-position *phoenix*)
                                         (+ (auv-y-position *phoenix*) 1) level))
                (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                         (- (auv-y-position *phoenix*) 1) level)
                (aref *area* (+ (auv-x-position *phoenix*) 1)
                                         (- (auv-y-position *phoenix*) 1) level))
                (setf (aref *auv-map* (auv-x-position *phoenix*)
                                         (- (auv-y-position *phoenix*) 1) level)
                    (aref *area* (auv-x-position *phoenix*)
                                        (- (auv-y-position *phoenix*) 1) level)))))
(8
    (case (auv-y-position *phoenix*)
        (0
            (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                         (auv-y-position *phoenix*) level)
                (aref *area* (- (auv-x-position *phoenix*) 1)
                                         (auv-y-position *phoenix*) level))
                (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                         (+ (auv-y-position *phoenix*) 1) level)
                    (aref *area* (- (auv-x-position *phoenix*) 1)
                                         (+ (auv-y-position *phoenix*) 1) level))
                (setf (aref *auv-map* (auv-x-position *phoenix*)
                                         (+ (auv-y-position *phoenix*) 1) level)
                    (aref *area* (auv-x-position *phoenix*)
                                         (+ (auv-y-position *phoenix*) 1) level)))
```

```
(8
        (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                        (auv-y-position *phoenix*) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                        (auv-y-position *phoenix*) level))
        (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                    (- (auv-y-position *phoenix*) 1) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                    (- (auv-y-position *phoenix*) 1) level))
        (setf (aref *auv-map* (auv-x-position *phoenix*)
                                    (- (auv-y-position *phoenix*) 1) level)
            (aref *area* (auv-x-position *phoenix*)
                                    (- (auv-y-position *phoenix*) 1) level)))
(otherwise
        (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                        (auv-y-position *phoenix*) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                        (auv-y-position *phoenix*) level))
        (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                    (+ (auv-y-position *phoenix*) 1) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                    (+ (auv-y-position *phoenix*) 1) level))
        (setf (aref *auv-map* (auv-x-position *phoenix*)
                                    (+ (auv-y-position *phoenix*) 1) level)
            (aref *area* (auv-x-position *phoenix*)
                                    (+ (auv-y-position *phoenix*) 1) level))
        (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                    (- (auv-y-position *phoenix*) 1) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                    (- (auv-y-position *phoenix*) 1) level))
        (setf (aref *auv-map* (auv-x-position *phoenix*)
                                    (- (auv-y-position *phoenix*) 1) level)
            (aref *area* (auv-x-position *phoenix*)
                                (- (auv-y-position *phoenix*) 1) level)))))
```

```
(otherwise
   (case (auv-y-position *phoenix*)
      (0
         (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level))
         (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                                  (+ (auv-y-position *phoenix*) 1) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                                  (+ (auv-y-position *phoenix*) 1) level))
         (setf (aref *auv-map* (auv-x-position *phoenix*)
                                                  (+ (auv-y-position *phoenix*) 1) level)
            (aref *area* (auv-x-position *phoenix*)
                                                  (+ (auv-y-position *phoenix*) 1) level))
         (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level)
            (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level))
         (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                  (+ (auv-y-position *phoenix*) 1) level)
            (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                  (+ (auv-y-position *phoenix*) 1) level)))
      (8
         (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level))
         (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                                  (- (auv-y-position *phoenix*) 1) level)
            (aref *area* (- (auv-x-position *phoenix*) 1)
                                                  (- (auv-y-position *phoenix*) 1) level))
         (setf (aref *auv-map* (auv-x-position *phoenix*)
                                                  (- (auv-y-position *phoenix*) 1) level)
            (aref *area* (auv-x-position *phoenix*)
                                                  (- (auv-y-position *phoenix*) 1) level))
         (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level)
            (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                  (auv-y-position *phoenix*) level))
         (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                  (- (auv-y-position *phoenix*) 1) level)
            (aref *area* (+ (auv-x-position *phoenix*) 1)
```

```
                                        (- (auv-y-position *phoenix*) 1) level)))
(otherwise
   (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                                (auv-y-position *phoenix*) level)
      (aref *area* (- (auv-x-position *phoenix*) 1)
                                                (auv-y-position *phoenix*) level))
   (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                        (- (auv-y-position *phoenix*) 1) level)
      (aref *area* (- (auv-x-position *phoenix*) 1)
                                        (- (auv-y-position *phoenix*) 1) level))
   (setf (aref *auv-map* (auv-x-position *phoenix*)
                                        (- (auv-y-position *phoenix*) 1) level)
      (aref *area* (auv-x-position *phoenix*)
                                        (- (auv-y-position *phoenix*) 1) level))
   (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                                (auv-y-position *phoenix*) level)
      (aref *area* (+ (auv-x-position *phoenix*) 1)
                                                (auv-y-position *phoenix*) level))
   (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                        (- (auv-y-position *phoenix*) 1) level)
      (aref *area* (+ (auv-x-position *phoenix*) 1)
                                        (- (auv-y-position *phoenix*) 1) level))
   (setf (aref *auv-map* (- (auv-x-position *phoenix*) 1)
                                        (+ (auv-y-position *phoenix*) 1) level)
      (aref *area* (- (auv-x-position *phoenix*) 1)
                                        (+ (auv-y-position *phoenix*) 1) level))
   (setf (aref *auv-map* (auv-x-position *phoenix*)
                                        (+ (auv-y-position *phoenix*) 1) level)
      (aref *area* (auv-x-position *phoenix*)
                                        (+ (auv-y-position *phoenix*) 1) level))
   (setf (aref *auv-map* (+ (auv-x-position *phoenix*) 1)
                                        (+ (auv-y-position *phoenix*) 1) level)
      (aref *area* (+ (auv-x-position *phoenix*) 1)
                                    (+ (auv-y-position *phoenix*) 1) level)))))))
```

```
; Implementation of AUVMINE in Lisp

; Author: Jose A. Rodrigues Nt.
; Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
; Date: September 1994
; Version: 0.5
; Compiler: Franz Allegro CL/PC version 1.0
;
; Function split-area: Divides the piece of the slice presently being sweept in two
; due to the existence of a wall or mountain. Call sweep-area recursivelly to each
; half of the chunk and also call the transit function to move the AUV between the
; two sub-areas.


(defun split-area (initial-waypoint final-waypoint)
    (let*
        ((north-limit (max (car initial-waypoint) (car final-waypoint)))
        (south-limit (min (car initial-waypoint) (car final-waypoint)))
        (up-limit (min (caddr initial-waypoint) (caddr final-waypoint)))
        (down-limit (max (caddr initial-waypoint) (caddr final-waypoint)))
        (sub-area-one-initial (list (auv-x-position *phoenix*) (auv-y-position *phoenix*)
                                                                (auv-z-position *phoenix*)))
        (sub-area-two-initial (nth (- (list-length *agenda*) 2) *agenda*))
        (sub-area-one-final (list
                                (if (plusp *x-motion*)
                                    (eval south-limit)
                                    (eval north-limit))
                                (auv-y-position *phoenix*)
                                (if (plusp *z-motion*)
                                    (eval down-limit)
                                    (eval up-limit))))
        (sub-area-two-final (list
                                (if (plusp *x-motion*)
                                    (eval north-limit)
                                    (eval south-limit))
                                (auv-y-position *phoenix*)
                                (if (plusp *z-motion*)
                                    (eval down-limit)
                                    (eval up-limit)))))
        (sweep-area sub-area-one-initial sub-area-one-final)
        (transit sub-area-two-initial)
        (sweep-area sub-area-two-initial sub-area-two-final)
        (setf (auv-status *phoenix*) 'sweeping-level)
```

```
(push (list (auv-x-position *phoenix*) (auv-y-position *phoenix*)
                                        (auv-z-position *phoenix*)) *agenda*)))
```

```
; Implementation of AUVMINE in Lisp

; Author: Jose A. Rodrigues Nt.
; Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
; Date: September 1994
; Version: 0.5
; Compiler: Franz Allegro CL/PC version 1.0
;
; Function transit: moves the auv from the end of one sub-area to the beginning of
;                   another.  May also be used later to take the AUV to any position,
;                   e.g., the search initial-position, if the AUV was not launched there.


(defun transit (destination)
   (do ((dummy (hill-path (list (auv-x-position *phoenix*) (auv-y-position *phoenix*)
                                (auv-z-position *phoenix*)) destination)))
      ;finds shortest path
      ((equal (list (auv-x-position *phoenix*) (auv-y-position *phoenix*)
                    (auv-z-position *phoenix*)) destination))
      ; check if the destination was reached
      (cond   ; if next waypoint is ok move and update map, else recalculate path
         ((next-safe)
            (move-to-waypoint (pop *sweep-path*))
            (look-around))
         (t (hill-path destination)))))
```

```
; Implementation of AUVMINE in Lisp

; Author: Jose A. Rodrigues Nt.
; Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
; Date: September 1994
; Version: 0.5
; Compiler: Franz Allegro CL/PC version 1.0
;
; Function check-wall: check the distance from the present AUV position to the position
;                      it  first encountered the obstacle. Returns true if the distance is
;                      bigger than 3 levels. Also checks if one of the area limits was
;                      reached. It is the decision function for split-area.

; Function large-obstacle: If the AUV is back to the search level where the go around
;                          procedure was initiated, check if the first sub-goal generated
;                          by the obstacle is in the back. If it is, pops it and all other
;                          sub-goal above from the agenda.

; Function left-back: Checks if the ongoing sub-goal was left behind.

(defun check-wall ()
    (if (> (list-length *agenda*) 1)
        (or (> (abs (- (auv-z-position *phoenix*)
                                    (caddr (nth (- (list-length *agenda*) 1) *agenda*)))) 3)
            (and (equal (auv-z-position *phoenix*) 1)
                 (minusp *z-motion*))
            (and (equal (auv-z-position *phoenix*) 9)
                 (plusp *z-motion*)))))


(defun large-obstacle ()
    (setq *agenda* (last *agenda*)))

(defun left-back (sub-goal)
    (plusp (/ (- (auv-x-position *phoenix*) (car sub-goal)) *x-motion*)))
```

```
;  Implementation of AUVMINE in Lisp

;  Author: Jose A. Rodrigues Nt.
;  Project: A Mine Searcher for AUV's - Master Thesis NPGS - 1994
;  Date: September 1994
;  Version: 0.5
;  Compiler: Franz Allegro CL/PC version 1.0
;
;  Function calc-prob: calculates the fraction of area searched after the search,
;                      based on the generated map.

(defvar *undetect-amount* 0)

(defun count-undetect (node)
    (let ((node-counter 0))
        (If (equal node 'unknown)
            (+ node-counter 1)
            (+ node-counter 0))))

(defun calc-prob (any-map)
    (setf *undetect-amount* 0)
    (loop
        with  x = 1
        and  y = 1
        and  z = 1
        for x from 1 to 9
            do (loop for  y from 1 to 9
                do (loop for z  from 1 to 9
                    do (setf *undetect-amount*
                        (+ *undetect-amount* (count-undetect (aref any-map x y z)))))))
    (- 1 (/  *undetect-amount* 721.0)))
```

## APPENDIX B. THE LISP SOCKETS SOURCE CODE

The following code implements the communications between the Lisp interpreter and the AUV Execution level. It is also used for communications between the Lisp interpreter and the Underwater Virtual World.

The code is based on the Commom Lisp Listener code, used to create the connection between Lisp and the EMACS editor. The program used by the Listener was modified to serve the AUV needs. Additional functions were created to perform the messaging passing between Lisp and the Execution Level. Two scripts showing the use of the functions are included.

```
;; Phoenix Lisp/Execution communications implementation

;; Author: Jose A. Rodrigues Nt.
; Project: Lisp sockets - Master Thesis NPGS - 1994
  File: os9_sock.cl
; Date: July 1994
; Compiler: Franz Allegro Commom Lisp version 4.0
;
;; Remarks: This code requires the Lisp-listener code (ipc.cl).


(load "mod_ipc.cl")
(use-package :ipc)
(load "init_os9.cl")
(load "get-os9.cl")
(load "send-os9.cl")
(load "end-os9.cl")
```

```
;; Phoenix Lisp/Execution communications implementation

;; Author: Jose A. Rodrigues Nt.
;  Project: Lisp sockets - Master Thesis NPGS - 1994
;  Date: July 1994
;  Compiler: Franz Allegro Commom Lisp version 4.0

;; Modified ipc.cl
;; These are the modifications needed for "ipc.cl" to work with the AUV
;; These  definitions must replace the definitons found in "ipc.cl" and the new file must be
;;  saved as "mod_ipc.cl".
;; The inet_addr function converts an internet address to an integer internet address.
;; Open-network-stream only accepts strings or integer addresses.


#-(version>= 4 0)
(export '(start-lisp-listener-daemon open-network-stream
          *inet-port-min* *inet-port-max* *inet-port-used* inet_addr))


(defparameter *inet-port-max* 3220
  "The largest internet service port number on which Lisp listens for
connections.")


(defparameter .needed-funcs.
        (mapcar #'convert-to-lang
            ;; this list appears in makefile.cl, too
            '("socket" "bind" "listen" "accept" "getsockname"
                "gethostbyname" "getservbyname"
                "connect" "bcopy" "bcmp" "bzero" "inet_addr")))
```

```
(eval-when (compile eval load)
        ;(unless .lisp-listener-daemon-ff-loaded.;; modified to allow inet_addr to load
            (excl::machine-case :host
            ((:apollo :tek4300 :rs6000))
            (t (unless (dolist (name .needed-funcs. t)
                 (if (not (entry-point-exists-p name))
                     (return nil)))
        (princ "; Loading TCP routines from C library...")
        (force-output)
        (unless (load ""
            :verbose nil
            :unreferenced-lib-names .needed-funcs.
             #+(target sgi4d) :system-libraries
            #+(target sgi4d) '("bsd")
            (error "foreign load failed"))
        (princ "done")
        (terpri)
        (force-output))))

        (setq .lisp-listener-daemon-ff-loaded. t)
        (defforeign-list '((getuid :entry-point #,(convert-to-lang "getuid"))
            (socket :entry-point #,(convert-to-lang "socket"))
            (bind :entry-point #,(convert-to-lang "bind"))
            (accept :entry-point #,(convert-to-lang "accept"))
            (getsockname :entry-point #,(convert-to-lang
                 "getsockname"))
            (gethostbyname :entry-point #,(convert-to-lang
                "gethostbyname"))
            (getservbyname :entry-point #,(convert-to-lang
                "getservbyname"))
            (select :entry-point #,(convert-to-lang "select"))
            (connect :entry-point #,(convert-to-lang "connect"))
            (bcopy :entry-point #,(convert-to-lang "bcopy"))
            (bzero :entry-point #,(convert-to-lang "bzero"))
            (bcmp :entry-point #,(convert-to-lang "bcmp"))
            (perror :entry-point #,(convert-to-lang "perror"))
            (unix-listen :entry-point #,(convert-to-lang "listen"))
            (unix-close :entry-point #,(convert-to-lang "close"))
            (inet_addr :entry-point #,(convert-to-lang "inet_addr")))
         :print nil));;)
```

```
(if* (entry-point-exists-p (convert-to-lang "lisp_htons"))
   then ;; Allegro CL 3.1 or later...
      (defforeign-list '((lisp_htons :entry-point #,(convert-to-lang
                              "lisp_htons"))
                  (lisp_htonl :entry-point #,(convert-to-lang
                              "lisp_htonl"))
                  (lisp_ntohs :entry-point #,(convert-to-lang
                              "lisp_ntohs"))
                  (lisp_ntohl :entry-point #,(convert-to-lang
                              "lisp_ntohl")))
      :print nil)
   else ;; pre-3.1 Allegro CL.  Do it the hard way...
      #+little-endian
      (progn
         (setf (symbol-function 'lisp_htons) #'(lambda (x)
            (logior (ash (logand x #x00ff) 8)
            (ash (logand x #xff00) -8))))
         (setf (symbol-function 'lisp_ntohs) #'(lambda (x) (lisp_htons x)))
         (setf (symbol-function 'lisp_htonl)  #'(lambda (x)
            (logior (ash (logand x #x000000ff)  24)
            (ash (logand x #x0000ff00)   8)
            (ash (logand x #x00ff0000)  -8)
            (ash (logand x #xff000000) -24))))
         (setf (symbol-function 'lisp_ntohl) #'(lambda (x) (lisp_htonl x))))
      #+big-endian
      (progn
         (setf (symbol-function 'lisp_htons) #'(lambda (x) x))
         (setf (symbol-function 'lisp_htonl) #'(lambda (x) x))
         (setf (symbol-function 'lisp_ntohs) #'(lambda (x) x))
         (setf (symbol-function 'lisp_ntohl) #'(lambda (x) x))))
```

```
;; Phoenix Lisp/Execution communications implementation

;; Author: Jose A. Rodrigues Nt.
; Project: Lisp sockets - Master Thesis NPGS - 1994
  File: init_os9.cl
; Date: July 1994
; Compiler: Franz Allegro Commom Lisp version 4.0
;
(defun init_os9 (machine)
        (defvar *os9* (open-network-stream :host machine :port 3210)))


;; get-os9 returns either the list fo characters in the input buffer
;; of the *os9* stream or nil if the buffer is empty.
;; pretty-get transforms the list of characters generated by get-os9
;; into a big string.
;;
;; Note: Everytime we read from the stream we get 81 characters.
;;       If the other side sent less than that, some trash will
;;       appear filling up the stream.

(defun get-os9 ()
        (if (listen *os9*)
            (let reply (setf reply nil)
            (dotimes (index 81 (nreverse reply))
            (setf reply (cons (read-char-no-hang *os9*) reply))))))

(defun pretty-get ()    ; returns a string
        (coerce (get-os9) 'string))
```

```
;; send-os9 writes a sequence of 81 characters to the *os9* stream.
;; Whatever is the size of your output string send-os9 will make it
;; 81 chars long. It will either fill up with junk or clip it if bigger
;; than 81.

;; Note: IMPORTANT -- The other side has always to read 81 chars or
;;       the communication will get out of synch. If one
;;       time the other side reads, for instance, 20 chars, 61 chars
;;       will be left over in the buffer and will appear on the next
;;       read. This will continue forever, unless you clear the buffer.
;;       To clear the buffer: (clear-output *os9*).

(defun send-os9 (command)
        (format *os9* "~81@<~A~>" command) ;sends command padded w/ blanks
        (force-output *os9*))           ;to complete 81 chars


(defun end-os9 ()
        (close *os9*))
```

```
Script started on Fri Oct 14 12:59:56 1994
{aquarius}/users/work3/rneto/auv/sockets/lisp: cl

Allegro CL 4.1 [SPARC; R1] (2/9/94 14:53)

;; Copyright Franz Inc., Berkeley, CA, USA
;; Unpublished.  All rights reserved under the copyright laws
;; of the United States.
;; Restricted Rights Legend
;; ------------------------
;; Use, duplication, and disclosure by the Government are subject to
;; restrictions of Restricted Rights for Commercial Software developed
;; at private expense as specified in DOD FAR 52.227-7013 (c) (1) (ii).
;; Optimization settings: safety 1, space 1, speed 1, debug 2
;; For a complete description of all compiler switches given the current
;; optimization settings evaluate (explain-compiler-settings).
user(1): (load "os9_sock.cl")
; Loading /work3/rneto/auv/sockets/lisp/os9_sock.cl.
;  Loading /work3/rneto/auv/sockets/lisp/mod_ipc.cl.
;  Loading /work3/rneto/auv/sockets/lisp/init_os9.cl.
;  Loading /work3/rneto/auv/sockets/lisp/get-os9.cl.
;  Loading /work3/rneto/auv/sockets/lisp/send-os9.cl.
;  Loading /work3/rneto/auv/sockets/lisp/end-os9.cl.
t
user(2): (j init_os9 "taurus")
*os9*
user(3): (pretty-get)
"SUCCESS #1:  os9server connected to os9sender!test handshake between hosts:
%s
"
user(4): (send-os9 "SUCCESS. Lisp connected to OS( 9) ")
nil
user(5): (pretty-get)
""
user(6): (pretty-get)
""
user(7): (send-os9 "WAYPOINT-HOVER 20 45 90 3) ")
nil
user(8): (PRETTY-GET        pretty-get)
"mine 203 45__·+ TüÇ  "
user(9): (end-os9)
t
script done
```

```
Script started on Fri Oct 14 12:59:50 1994
{taurus}/users/work3/rneto/auv/sockets/don: os9server -t
[os9server TRACE on]

os9server socket 'open' successful
os9server socket 'bind' successful
os9server socket 'listen' successful ...
os9server socket waiting to 'accept' ...

os9server connection is open between networks.
os9server SERVER: socket_descriptor = 3,
            socket_accepted   = 4,
            socket_stream     = 4
test handshake between hosts:

SUCCESS. Lisp connected to OS9
                        □0X

mine 203 45
os9server receiver block loop bytes_read = 81

WAYPOINT-HOVER 20 45 90 3
[os9server command_sent:mine 203 45]
os9server send_telemetry_to_server loop  bytes sent = 81
os9server send_telemetry_to_server total bytes sent = 81

os9server receiver block loop bytes_read = 0
os9server get_PDU_from_other_host read received 0 bytes

(shutdown)        shutdown
[os9server command_sent:shutdown]
os9server send_telemetry_to_server loop  bytes sent = 81
os9server send_telemetry_to_server total bytes sent = 81
os9server shutdown in progress ...
os9server shutdown_os9server () complete
os9server shutdown in progress ...
os9server close (socket_stream) failed
os9server shutdown_os9server () complete
os9server exit.

{taurus}/users/work3/rneto/auv/sockets/don: exit
script done
```

**INITIAL DISTRIBUTION LIST**

Defense Technical Information Center 2
Cameron Station
Alexandria, VA 22304-6145

Library, Code 52 2
Naval Postgraduate School
Monterey, CA 93943-5002

Chairman, Code CS 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Chairman, Code OR 1
Operations Research Department
Naval Postgraduate School
Monterey, CA 93943

Dr. Gordon H. Bradley, Code OR/Bz 2
Operations Research Department
Naval Postgraduate School
Monterey, CA 93943

Dr. Robert B. McGhee, Code CS/Mz 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Dr. James N. Eagle, Code OR/Er 1
Operations Research Department
Naval Postgraduate School
Monterey, CA 93943

Dr. Anthony J. Healey, Code ME/Hy 1
Chairman, Mechanical Engineering Department
Naval Postgraduate School
Monterey, CA 93943

Dr. Alan R. Washburn, Code OR/Ws 1
Operations Research Department
Naval Postgraduate School
Monterey, CA 93943

Comando de Operacoes Navais 1
Attn: Brazilian Naval Commission
4706 Wisconsin Ave., N.W.
Washington, DC 20016

Centro de Analises de Sistemas Navais 1
Attn: Brazilian Naval Commission
4706 Wisconsin Ave., N.W.
Washington, DC 20016

LCDR Donald P. Brutzman, Code OR/Br 1
Operations Research Department
Naval Postgraduate School
Monterey, CA 93943

LCDR Almir Garnier Santos 1
Attn: Brazilian Naval Commission
4706 Wisconsin Ave., N.W.
Washington, DC 20016

Lt Jose A. Rodrigues Neto 2
Attn: Brazilian Naval Commission
4706 Wisconsin Ave., N.W.
Washington, DC 20016